Tito Miguel de Sousa Ribeiro Martins Ferreira

Licenciado em Engenharia Informática

Extensão do NoHR para a web

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Matthias Knorr, Professor Auxiliar,
Universidade NOVA de Lisboa

Júri

Presidente: Henrique João Lopes Domingos
Vogais: Vítor Manuel Beires Pinto Nogueira
Jorg Matthias Knorr

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Abril, 2020

Extensão do NoHR para a Web

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Agradecimentos

Queria deixar uma pequena dedicatória nesta dissertação a agradecer a todos aqueles que me apoiaram direta e indiretamente nesta etapa da minha vida e me ajudaram a ultrapassar todas as dificuldades que surgiram, sem eles não teria conseguido concluir este projeto que marca o fim de uma etapa muito importante da minha vida.

Gostaria de agradecer a todos os professores, em especial ao meu coordenador, o professor Matthias Knorr, por ter estado sempre presente na realização deste projeto e por estar sempre disponível para esclarecer qualquer questão. A disponibilidade para mim e todo o apoio que me deu durante a realização desta dissertação foram sem dúvida nenhuma uma peça fundamental para a conclusão desta etapa de aprendizagem e desenvolvimento pessoal.

Gostaria também de deixar umas palavras de agradecimento à minha família e amigos pela paciência que tiveram comigo e por nunca me deixarem desmotivar durante este percurso. O seu apoio foi sem dúvida muito importante para mim e sem ele não teria sido possível ultrapassar este desafio.

Em especial, gostaria de agradecer profundamente aos meus pais e à minha namorada, por terem uma paciência inigualável para comigo e por me incentivarem intensivamente a trabalhar neste projeto, sem eles não teria sido possível concluir esta etapa marcante da minha vida.

Resumo

A integração de ontologias, que empregam o formalismo de mundo aberto, com regras, que empregam o formalismo de mundo fechado, não é de todo uma tarefa trivial, embora esta integração seja fundamental para certas aplicações na prática. Aplicações por exemplo da área da saúde, onde, por vezes é necessário que o conhecimento esteja explicitamente presente de forma a que se possam tirar conclusões (formalismo de mundo aberto) e noutros casos, a ausência desse conhecimento é suficiente para se poder tirar uma conclusão (formalismo de mundo fechado), no qual ambos os formalismos são necessários e, portanto, a sua integração é essencial para se poderem tirar conclusões adequadas.

Estando a par destas questões, o raciocinador híbrido NoHR que é um plug-in para o editor de ontologias *Protégé*, permite a integração de uma ontologia escrita em qualquer um dos perfis do OWL 2 com um conjunto de regras não-monotónicas, respondendo a perguntas usando uma estratégia *top-down*, que significa que apenas a parte da ontologia e das regras que é relevante para a pesquisa é analisada, melhorando assim o desempenho.

Neste momento, para utilizar o NoHR é necessária uma instalação do *Protégé* e posteriormente, configurar o NoHR e instalar o motor de regras XSB Prolog que permite executar as pesquisas. Todo este trabalho tem de ser feito por cada utilizador em cada computador onde se pretenda usar o plug-in.

Com a grande adesão a plataformas web 2.0 e a adoção gradual de ontologias e tecnologias da web semântica em contextos práticos, são então necessárias novas ferramentas de desenvolvimento de ontologias que sejam mais adequadas às novas formas de interação, construção e consumo de conhecimento. Para solucionar esta limitação, pretende-se desenvolver uma aplicação web que facilite a utilização no NoHR, fornecendo uma ferramenta acessível por qualquer navegador de web, deixando de ser necessário a instalação e configuração do NoHR no *Protégé*.

O NoHR Web, será a primeira aplicação web que integra ontologias com regras não-monotónicas, onde não serão necessárias instalações. Desta forma todo o trabalho dos utilizadores ficará nas sua contas e facilitará o trabalho em diferentes computadores.

Palavras-chave: Web Semântica, Ontologias, Regras, NoHR, Aplicação Web

Abstract

The integration of ontologies, which employ the open-world formalism, with rules, which employ closed-world formalism, is not a trivial task at all, though this integration is fundamental to certain aplications in practice. Applications, for example, in clinical health care, where it is sometimes necessary for knowledge to be explicitly present in order to draw conclusions (open world formalism) and in other cases, the absence of such knowledge is sufficient to draw a conclusion (closed-world formalism), so, both formalisms are necessary and therefore their integration is essential to take adequate conclusions.

Being aware of this matter, the hybrid reasoner plug-in NoHR for the ontology editor *Protégé* that integrates an ontology (OWL 2) with non-monotonic rules using a top-down approach, which means that only the part of the ontology and rules that is relevant for the query is actually evaluated, improving performance this way.

As a *Protégé* plug-in, NoHR currently requires a *Protégé* instalation and, setting up NoHR and a installation of a rule engine XSB Prolog to answer queries. All this work must to be done by each user on each computer where the plugin is to be used.

With the wide adoption of Web 2.0 platforms and the gradual adoption of ontologies and Semantic Web technologies in practice, we need ontology-development tools that are better suited for the novel ways of interacting, constructing and consuming knowledge. To solve this limitation, we develop a web application, which simplifies working with NoHR considerably, providing a tool that is accessible from any Web browser, no longer requiring the installation and the setup of NoHR in *Protégé*.

With a web application of NoHR, this will be the first web application that integrates ontologies with non-monotonic rules, which does not require any instalation, allows the users to store their work in their accounts, and facilitates working on different computers.

Keywords: Semantic Web, Ontology, Rules, NoHR, Web application

Índice

Lista de Figuras

Siglas

AJAX Asynchronous JavaScript and XML.

API Application Programming Interface.

BSON Binary JSON.

CWA Close World Assumption.

DBMS Database Management System.

DL Description Logics.

GHz Gigahertz.

GWT Google Web Toolkit.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

IRI Internationalized Resource Identifier.

JSON JavaScript Object Notation.

KRR Knowledge Representation and Reasoning.

MKNF Minimal Knowledge and Negation as Failure.

NoHR Nova Hybrid Reasoner.

ODBC Open Database Connectivity.

OWA Open World Assumption.

OWL Web Ontology Language.

POM Project Object Model.

RAM Random Access Memory.

RDF Resource Description Framework.

REST Representational State Transfer.

RIF Rule Interchange Format.

RPC Remote Procedure Call.

SDK Software Development Kit.

SKOS Simple Knowledge Organization System.

SOAP Simple Object Access Protocol.

SPARQL Protocol and RDF Query Language.

SQL Structured Query Language.

SWRL Semantic Web Rule Language.

W3C World Wide Web Consortium.

XML Extensible Markup Language.

Capítulo 1

Introdução

Neste capítulo vamos fazer uma introdução à web semântica e explicar o porquê da sua importância. Vamos também introduzir o plug-in NoHR, assim como as suas limitações, onde vamos abordar o plano de trabalhos inclusivamente.

1.1 Motivação da web semântica

A visão da web semântica pode ser resumida na seguinte frase: *Tornar a web mais acessível para os computadores*. No princípio, a web era uma web de texto e imagens, o que é bastante útil para as pessoas, mas onde os computadores têm um papel bastante limitado. Os computadores apenas indexam palavras-chave, devolvem informações dos servidores para os clientes, e nada mais. Era assim na web clássica, em que todo o trabalho inteligente como selecionar, combinar e agregar era essencialmente feito pelo utilizador, como referido em [2].

E se pudéssemos tornar a web mais rica para os computadores, onde a informação adicionada pudesse ser lida e compreendida pelas máquinas. Uma web com esta estrutura tornaria possível muitas tarefas que atualmente são impossíveis na web clássica. Por exemplo, fazer uma pesquisa não seria limitado apenas a pesquisar por palavras-chave, deveria ser mais eficiente e preciso onde a pesquisa podia incluir os sinónimos, ter atenção aos antónimos e ter em conta o contexto e o propósito da pesquisa.

Atualmente a ideia base da web semântica é estender as páginas web com conteúdo estruturado que possa ser interpretado pelos computadores, onde a utilização de técnicas de raciocínio e representação de conhecimento (KRR) são usadas para realçar a informação na *World Wide Web*, permitindo que a informação disponível seja processada por sistemas inteligentes.

Hoje em dia, a informação na web está muito melhor organizada para os computadores, onde os mesmos têm um papel muito mais importante na análise e interpretação da informação disponível.

1.2 Ontologias

Na web semântica, os formalismos de KRR são ontologias que definem conceitos e relações (também referidos como "termos") usadas para descrever uma área de interesse. As ontologias são usadas para classificar os termos que podem ser utilizados em algum caso específico, caracterizar possíveis relações, e definir restrições de uso sobre os termos. As ontologias podem ser bastante complexas, como o Snomed CT[1] ou o Galen[2], que são ontologias da área da saúde que contêm milhares de termos, ou bastante simples, descrevendo apenas um ou dois conceitos.

O papel das ontologias na web semântica é facilitar a integração da informação, por exemplo, quando existem ambiguidades em relação a termos usados para definir diferentes conjuntos de informação, ou quando um fragmento extra de conhecimento permite levar à descoberta de novas relações. Considerando o exemplo das ontologias no campo dos cuidados de saúde. Os médicos utilizam-nas para representar o conhecimento sobre sintomas, doenças e tratamentos. As empresas farmacêuticas usam-nas para representar conhecimento sobre medicamentos, dosagens, alergias, entre outros. Combinando este conhecimento das comunidades médicas e farmacêuticas com a informação dos pacientes permite abrir uma porta a uma vasta gama de aplicações como ferramentas de suporte de decisão que procurem por possíveis tratamentos, sistemas que avaliem a eficiência de medicamentos e os seus possíveis efeitos secundários e ferramentas que apoiem a pesquisa epidemiológica, como referido em [3].

Outro exemplo da utilização de ontologias é o uso na organização de conhecimento, como em bibliotecas, museus, jornais, artefactos históricos, entre outros, podem usar ontologias, usando formalismos, para potenciarem o poder da *linked data* [4].

Depende das aplicações a complexidade das ontologias que pretendem usar, em que algumas podem decidir usar ontologias grandes e complexas, e depender da lógica da aplicação. Outras aplicações podem preferir usar ontologias mais simples, e deixar o ambiente da web semântica usar a informação extra para fazer a identificação dos termos.

Para satisfazer estas necessidades, o *World Wide Web Consortium* (W3C)[5] oferece uma vasta gama de linguagens estandardizadas para descrever e definir diferentes formas de ontologias num formato padrão. Estas incluem *RDF and RDF Schema* [10], *Ontology Web Language* [22] (OWL), e *Rule Interchange Format* [18] (RIF), embora o RIF defina uma norma que permite a troca de regras entre os sistemas de regras, e não uma linguagem

[1] http://www.snomed.org/
[2] https://bioportal.bioontology.org/ontologies/GALEN
[3] https://www.w3.org/standards/semanticweb/ontology
[4] https://www.w3.org/standards/semanticweb/data.html
[5] https://www.w3.org/

que sirva todos os propósitos como as primeiras três mencionadas. A escolha entre estas diferentes tecnologias depende dos objetivos que uma dada aplicação pretende.

De seguida vamos fazer uma introdução ao RDF, ao OWL 2, e ao RIF onde vamos apresentar as principais características de cada linguagem.

1.2.1 RDF e RDF Schema

O *Resource Description Framework* (RDF) [10] define um modelo de dados para a web semântica, permitindo a junção de informação, mesmo que os esquemas subjacentes sejam diferentes.

O RDF estende a estrutura de ligações da web ao usar URIs para denominar as relações entre conceitos, assim como nas pontas da ligação (denominada como triplo), como no caso do Simple Knowledge Organization System (SKOS) [23], que é uma aplicação nesta base. A utilização deste modelo permite que a informação estruturada e semi-estruturada possa ser misturada e partilhada entre diferentes aplicações.

Esta estrutura de ligações forma grafos etiquetados e orientados, onde as arestas representam as ligações entre dois recursos, representados por nós no grafo.

A ontologia RDF Schema é uma linguagem de descrição de ontologia para descrever propriedades e classes de recursos RDF, com uma semântica de generalização de hierarquias de classes e propriedades, onde também é possível definir o domínio e o alcance (*range*) das propriedades.

1.2.2 OWL 2

A W3C Web Ontology Language (OWL) [22] é uma linguagem da web semântica desenhada para representar o conhecimento complexo sobre matérias, grupos de matérias, e relações entre essas matérias.

A versão atual do OWL, o OWL 2 [11] é uma linguagem mais expressiva do que o RDFS, no sentido em que também contém o significado de classes e propriedades, e as suas diferenças estão no facto de conter primitivas que suportam uma maior expressividade.

Mas apenas estender o RDFS leva a um raciocínio ineficiente e contra a obtenção do poder expressivo. Os construtores do RDFS como o *rdfs:Class* (classe de todas as classes) e *rdfs:Property* (classe de todas as propriedades) são muito expressivos e levam à perda das propriedades computacionais se a lógica do OWL 2 incluir estas primitivas.

É impossível obter raciocínio eficiente para uma linguagem tão expressiva como a combinação do RDFS com uma lógica tão poderosa.

Esta questão levou a que a linguagem OWL 2 fosse dividida em duas sub-linguagens, o OWL 2 Full: RDF-Based Semantics e o OWL 2 DL: Direct Semantics, cada uma com o intuito de resolver diferentes aspetos. A primeira, usa todas as primitivas da linguagem OWL 2, enquanto a segunda é mapeada para uma lógica de descrição, nomeadamente, a lógica de descrição $SROIQ$ [13] subjacente à norma W3C. As lógicas de descrição são

sub-conjuntos da lógica de primeira ordem onde o raciocínio eficiente é possível, através da perda de expressividade da linguagem.

Como a lógica de descrição $SROIQ$ [13] é bastante expressiva e geral, foram definidos três perfis que são mais restritivos que o OWL 2 DL, onde a semântica da sintaxe dos perfis é dada pela especificação OWL 2 DL. No capítulo 2, iremos analisar esta questão mais detalhadamente.

1.3 Regras

O formalismo de regras existe há várias décadas, com uma grande variedade de regras, por exemplo, regras de ação, regras de primeira ordem e programação lógica. Como consequência, o objetivo do *W3C Rule Interchange Format Working Group* [18] não é desenvolver uma nova linguagem de regras que sirva todos os propósitos, mas sim, focar-se no intercâmbio entre os vários sistemas de regras na web.

Esta abordagem levou ao desenvolvimento de uma família de linguagens, chamadas dialetos. O RIF definiu dois tipos de dialetos: os *Logic-based dialects* e os *Rules with actions*. Geralmente, os *Logic-based dialects* incluem linguagens que aplicam algum tipo de lógica, como a lógica de primeira ordem (normalmente restringida à lógica de Horn) ou lógica que não é de primeira ordem subjacente a várias linguagens de programação de lógica, como as lógicas de programação subjacente às semânticas *well-founded* [5] ou *stable* [6].

Concretamente, os dialetos definidos sobre os *Logic-based dialects* são:

- *RIF Core*, que corresponde essencialmente à lógica de Horn sem símbolos de funções (*function-free*, normalmente chamada "Datalog")
- *RIF Basic Logic Dialect (BLD)*, que corresponde essencialmente à lógica de Horn com igualdade

No caso das *Rules with actions*, devem incluir sistemas de produção e regras reativas. O dialeto desenvolvido para este ramo é o *Production Rule Dialect (RIF-PRD)*.

Nas ontologias, de forma a ser possível tirar conclusões "negativas"(que contrariem conhecimento já adquirido), é necessário que essa informação esteja explicitamente presente nas bases de conhecimento, pelo que, com a introdução de regras (não-monotónicas) é possível tirar outro tipo de conclusões em relação à informação que está presente nas bases de conhecimento, trazendo assim a noção do formalismo de mundo fechado (CWA).

As regras permitem expressar relações que não são em árvore [34] [6] como por exemplo, "um tio é o irmão de um pai"; restrições de integridade [28] do estado, ex. que certa informação está explicitamente presente na bases de dados; e formalismo de mundo fechado e especificação de exceções, como referido em [19].

[6] A DL SROIQ [13] também permite expressar axiomas de *role composition*, que podem ser usados para endereçarem (não todos) casos de uso

Por outro lado, as ontologias permitem expressar o formalismo de mundo aberto, em que é possível expressar por exemplo, "todas as pessoas têm um pai e uma mãe e ambos são pessoas". Nos formalismos baseados em regras não é possível concretizar a afirmação do exemplo sem listar todos os “pais” explicitamente.

1.4 Integração de ontologias com regras

Como mencionado anteriormente, as ontologias são um formalismo da norma de mundo aberto (OWA) enquanto as regras aplicam normalmente a norma de mundo fechado (CWA). Contudo a integração de ontologias com regras não é uma tarefa trivial, visto que esta integração pode ser indecidível [12]. Formalismos de regras e formalismos de ontologias diferem substancialmente na forma como obtêm a decidibilidade. Nas ontologias, a decidibilidade é obtida ao especificar restrições sintáticas nos predicados da lógica de primeira ordem disponíveis, e ao restringir a forma como esses predicados se relacionam. As regras, por sua vez, não têm restrições sintáticas, mas são geralmente limitadas na forma como os diferentes objetos aparecem nas bases de conhecimento, como referido em [19].

A decisão de usar o formalismo OWA parece natural no âmbito de aplicações relacionadas com a *World Wide Web* em que a falta de um pedaço de conhecimento nem sempre deve ser considerado como sendo falso. Embora existam cenários onde o formalismo CWA pareça mais natural, em sistemas de bases de dados normalmente assumimos que o conhecimento é completo e portanto qualquer conhecimento que não esteja contemplado na base de dados é assumido como falso.

Como exemplo, onde a combinação dos formalismos de OWA e CWA são benéficos, podemos considerar o grande caso de estudo descrito em [27], que contém milhões de asserções sobre os registos dos pacientes correspondentes com os critérios de ensaios clínicos. Neste domínio, o raciocínio sobre o mundo aberto é necessário em radiologia e dados de laboratório. A não ser que um teste dê explicitamente negativo, nenhuma conclusão pode ser tirada. Isto é, apenas podemos concluir que um paciente não tenha nenhuma espécie de cancro, se concluirmos que o teste deu negativo.

Por outro lado, o formalismo de mundo fechado também pode ser usado sobre dados de tratamento médico para avaliar se um paciente está ou não sobre medicação. Neste caso, com a ausência de conhecimento que informe que um paciente está a receber um certo tratamento, é possível concluir que o mesmo não está a receber esse tratamento.

Com este exemplo podemos ver claramente a importância e a necessidade da integração dos formalismo OWA e CWA para este domínio.

1.5 Introdução ao NoHR

O NoHR [21] (Nova Hybrid Reasoner) é um plug-in para o editor de ontologias *Protégé* [7] que permite que os utilizadores façam pesquisas integrando uma ontologia e um sistema de regras não-monotónicas.

O NoHR tem por base teórica o formalismo o MKNF Híbrido sobre *Well-founded semantics* [19], que tem dois argumentos importantes a seu favor (Comparando com o trabalho relacionado [4] [24] sobre combinar lógicas de descrição com regras não-monotónicas).

O primeiro argumento, uma abordagem geral, introduzido em [24] e baseada na lógica de *minimal knowledge and negation as failure* (MKNF) [20], fornece uma *framework* geral e flexível para combinar lógicas de descrição com regras não-monotónicas (ver em [24]).

O segundo argumento [19], que é uma variante de [24] baseado na *Well-founded semantics* [5] para programas de lógica, tem uma complexidade inferior ao anterior, sendo polinomial dentro das lógicas de descrição polinomiais e usa uma estratégia *top-down* nas pesquisas, assim como o SLG(O) [1], respondendo às perguntas baseando-se apenas na informação relevante para a pesquisa, sem ter a necessidade de computar o modelo inteiro. Esta característica é bastante útil em sistemas que contêm grandes ontologias com grandes quantidades de informação.

O NoHR integra as capacidades dos raciocinadores das lógicas de descrição ELK [17], HermiT [7], e o Konclude [31] com o sistema de regras XSB Prolog [8], contando também com as seguinte funcionalidades, mencionadas em [21].

- Permite que as ontologias sejam escritas em qualquer um dos três perfis tratáveis do OWL 2, e permite ainda combinar todos os construtores permitidos;
- Permite ainda um número vasto de predicados padrão para o Prolog;
- Editor de regras no *Protégé* acompanhado com um analisador de regras para corresponder às novas extensões;
- Possibilidade de definir predicados com um número arbitrário de argumentos no *Protégé*;
- Garante que as respostas às pesquisas terminam;
- Robustez entre as ontologias e as regras;
- Tempos de resposta interativos rápidos;

Como hoje em dia grande parte da informação é guardada em sistemas de bases de dados, os utilizadores do NoHR têm de fazer o trabalho de seleção e recolha dos dados manualmente, para então depois carregarem o ficheiro de regras e da ontologia para o

[7] https://protege.stanford.edu
[8] http://xsb.sourceforge.net

NoHR . Uma das desvantagens desta aproximação, para além de ser um trabalho que tem de ser feito pelo utilizador, é o tempo de processamento e o consumo de memória.

Para colmatar esta limitação, o NoHR foi estendido para integrar regras, ontologias e bases de dados através de mapeamentos [16]. Os mapeamentos representam uma ponte que conecta a extensão do formalismo híbrido com resultados de pesquisas SQL.

A última versão do NoHR (4.0) é baseada na extensão híbrida de bases de conhecimento, onde os mapeamentos são implementados usando *drivers* ODBC. Como resultado, o raciocinador suporta a integração com todos os maiores sistemas de bases de dados.

1.6 Limitações do NoHR

Apesar do *Protégé* permitir a instalação de plug-ins, o utilizador está sempre limitado ao poder de processamento da sua máquina, e no caso do utilizador pretender partilhar um projeto com outro utilizador, terá de passar a informação toda para ficheiros e enviar esses ficheiros para outro utilizador, não permitindo que uma pessoa possa trabalhar em diferentes dispositivos ou partilhar projetos de uma forma eficiente, sendo que é também necessária uma instalação e configuração do *Protégé* e do NoHR antes de se poder utilizar.

É necessária a instalação do motor de regras XSB Prolog que permite executar as pesquisas e o utilizador pode também instalar o Konclude para lidar com pesquisas em grandes ontologias, sendo que a instalação do Konclude é opcional. Todas estas instalações e configurações tornam o processo de configuração do NoHR bastante extenso e complexo.

Neste momento, existe uma versão web do *Protégé* denominado *WebProtégé* [9], mas esta versão também tem certas limitações.

O *WebProtégé* permite que qualquer utilizador possa criar uma conta e utilizar a aplicação livremente, permitindo que os utilizadores possam criar ou carregar um projeto. O facto do *WebProtégé* ser uma aplicação web facilita bastante a sua utilização, onde todos os projetos ficam associados à conta do utilizador o que permite que se trabalhe de qualquer dispositivo sem ser necessária uma configuração prévia como acontece no *Protégé*.

Contudo, o *WebProtégé* não permite a instalação de plug-ins por parte de terceiros e portanto neste momento não é possível integrar o *plug-in* NoHR no *WebProtégé*. Devido a isso, todas funcionalidades que o NoHR trazia ao *Protégé* não pode trazer ao *WebProtégé* visto que, por norma, o *WebProtégé* não integra ontologias com regras não-monotónicas.

1.7 Contribuições

Com o intuito de expandir as funcionalidades do NoHR para a web, foi desenvolvida uma aplicação web para integrar o *plug-in* NoHR , que torne o desenvolvimento de ontologias mais adequado às novas formas de interação, construção e consumo de conhecimento,

[9] https://webprotege.stanford.edu

como referido em [33], deixando então de ser necessário instalar o editor de ontologias e fazer toda a configuração de instalação do NoHR .

Com o NoHR Web, os utilizadores do NoHR podem migrar todo o seu trabalho para a web, que é acessível por qualquer *browser* da web, em qualquer dispositivo com acesso à Internet, onde todo o esforço de computação passa a estar do lado do servidor, retirando assim esse esforço de computação do cliente.

Outra característica inerente ao NoHR Web, é a possibilidade de partilha de projetos, possibilitando que vários utilizadores possam aceder às funcionalidades do NoHR em simultâneo no mesmo projeto. Será introduzida a estrutura desenvolvida para permitir que as funcionalidades do NoHR possam integrar um servidor na Web. Para além disso, no NoHR Web todo o trabalho do utilizador fica associado à sua conta na aplicação, retirando desta forma, a responsabilidade ao utilizador de guardar a informação do projeto no final da sua sessão.

Posteriormente, iremos introduzir a interface da aplicação web, que foi criada de raiz para o NoHR Web, onde serão apresentadas todas as janelas que integram a interface e será explicada a forma como os utilizadores poderão interagir com a aplicação web.

Por fim, iremos apresentar os testes efetuados ao NoHR Web, de forma a fazer uma comparação com o NoHR *desktop* e tirar conclusões de eficiência. Foram realizados testes de carregamento de ontologias em ambas as aplicações, onde foi comparado o tempo de carregamento. Com o intuito de avaliar a eficiência do processo de raciocínio, foram também analisados os tempos de inicialização do raciocinador do NoHR nas duas aplicações.

No caso do NoHR Web, foram feitos testes de sobrecarga onde se utilizaram vários utilizadores a executar pedidos em simultâneo.

Dito isto, no capítulo 2, serão apresentados mais em detalhe os temas introduzidos neste capítulo. No capítulo 3, será introduzida a componente do servidor do NoHR Web, em que vamos analisar com mais detalhe os pontos mencionados anteriormente. No capítulo 4, detalharemos a interface da aplicação em que serão analisados em detalhe todos os novos elementos introduzidos na interface. Por fim, no capítulo 5, apresentaremos os testes e analisaremos os resultados efetuados ao NoHR Web.

Com o NoHR Web, não só colmatamos todas as limitações referidas anteriormente, como também vai ser a primeira aplicação web que integra ontologias com regras não-monotónicas, o que é algo inovador que traz bastantes benefícios a esta área.

Capítulo 2

Estado da arte

Neste capítulo vamos falar mais em detalhe sobre ontologias e os tipos de regras e abordar o estado da arte relativamente ao plug-in NoHR e à aplicação WebProtégé, assim como será feita uma introdução ao trabalho realizado para criar a versão web do NoHR.

2.1 Ontologias

Tipicamente, uma ontologia consiste numa lista finita de termos e nas relações entre esses mesmos termos. Estes termos denotam conceitos importantes (classes de objetos) do domínio. Por exemplo, num ambiente universitário, os funcionários, estudantes, professores, cursos e disciplinas são alguns dos conceitos importantes.

As ontologias podem incluir uma hierarquia de classes, ou seja, uma propriedade de entre duas classes. Uma hierarquia de classes especifica que uma classe C é uma subclasse de uma outra classe C' se todos os objetos de C também estiverem incluídos em C'. Por exemplo, todos os secretários são funcionários.

Para além de hierarquias de classes, as ontologias incluem informação como:

- propriedades (X ensina Y)
- restrições de valores (apenas os professores podem dar aulas)
- disjunções (estudantes e funcionários são classes disjuntas)
- especificação de relações lógicas entre os objetos (cada departamento deve incluir pelo menos 10 professores)

No contexto da web, as ontologias fornecem uma compreensão partilhada de um domínio. Este tipo de compreensão partilhada é essencial para ultrapassar diferenças

na terminologia. Uma aplicação pode usar a nomenclatura *zipcode* enquanto uma outra aplicação usa *postcode* para descrever a mesma classe, como referido em [2].

Outro problema pode ser o facto de duas aplicações usarem o mesmo termo mas com significados diferentes. Estas diferenças podem ser ultrapassadas ao mapear uma dada ontologia para uma ontologia partilhada ou definindo mapeamentos entre ambas as ontologias. Em qualquer um dos casos, é possível observar que as ontologias suportam interoperabilidade semântica.

As ontologias são bastante úteis para as organizações e para a navegação dentro de uma página web, visto que, muitas páginas web expõem uma barra que contem níveis numa hierarquia de conceitos de termos.

Além disso, as ontologias são úteis para melhorar a precisão das pesquisas na web. Uma pesquisa pode procurar as páginas que se referem precisamente ao conceito de uma ontologia, em vez de devolverem todas as páginas em que alguma palavra-chave seja igual à pesquisada.

Para preencher todas as necessidades, existem diversas técnicas para descrever e definir ontologias num formato *standard*, como o *RDF e RDF Schema*, *Simple Knowledge Organization System (SKOS)*, *Web Ontology Language (OWL)* e *Rule Interchange Format (RIF)*. A escolha de um destes formatos em vez de outro, depende da complexidade pretendida para uma certa aplicação.

De seguida vamos falar mais em detalhe das linguagens ontológicas RDF e o OWL 2.

2.2 RDF e RDF Schema

O RDF [10] é uma linguagem universal que permite que os utilizadores descrevam recursos usando o seu próprio vocabulário. O RDF não faz suposições sobre qualquer domínio aplicacional, nem define semânticas para qualquer domínio. De forma a especificar uma semântica, o utilizador deve definir o significado do seu vocabulário em termos de um conjunto de estruturas independentes de domínio básico definidos pelo RDF Schema.

Fundamentalmente, o RDF é composto por recursos, propriedades, declarações e grafos, como referido em [2].

- **Recursos** são algo sobre o qual se pretende falar, ou melhor, são os conceitos que pretendemos definir e especificar, por exemplo, pessoas, lugares, países, cidades, entre outros. Cada recurso tem um URI associado que o identifica. Um URI pode ser um URL ou outro tipo de identificador único.

- **Propriedades** são um tipo especial de recurso que descrevem relações entre recursos, por exemplo, "localizado em", "realizado por", entre outros. As propriedades também são identificadas por URI.

- **Declarações** são triplos de entidade-atributo-valor, mais especificamente, um recurso, uma propriedade e um valor, respetivamente. Em que no caso do valor, tanto

```
@prefix : <http://foo.ex/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema>
:person1 :name "John";
:person1 :drives :car1 .
:person2 :name "Mary";
:person2 :drives :car2 .
```

Listagem 2.1: Exemplo de triplos em RDF

```
:John rdf:type :Person .
```

Listagem 2.2: Exemplo de asserção em OWL 2

```
_:x rdf:type owl:Class ;
owl:unionOf ( :Man :Woman ) .
```

Listagem 2.3: Exemplo de expressão em OWL 2

pode ser um recurso como um literal, sendo que os literais são valores atómicos, por exemplo, números, strings ou datas.

- **Grafos** são completamente compatíveis com a estrutura de triplos, é possível converter qualquer triplo para o formato de grafos.

Na listagem 2.1, temos um exemplo que contém a definição de triplos em RDF, e que demonstra como é que através de relações é possível especificar os conceitos sobre os quais pretendemos falar.

Onde : $person1$, : $person2$, : $car1$ e : $car2$ são recursos, que neste caso são pessoas e carros e que é conhecimento sobre o qual queremos falar. Como propriedades temos : $name$ e : $drives$ que descrevem a relação entre os recursos já mencionados. Neste exemplo temos também literais que são valores atómicos, como por exemplo, $"John"$ e $"Mary"$.

2.3 OWL 2

Comparativamente com o RDF, o OWL 2 é uma linguagem muito mais expressiva e permite definir informação mais complexa.

Em OWL 2, os membros de classes são normalmente chamados de indivíduos. Quando definimos que um indivíduo é de um certo tipo, estamos a declarar uma asserção, como observado na listagem 2.2.

Quando juntamos classes, propriedades e indivíduos, formamos expressões, como observado na listagem 2.3.

```
:Person owl:equivalentClass _:x .
_:x rdf:type owl:Class ;
owl:unionOf ( :Man :Woman ) .
```

Listagem 2.4: Exemplo de axioma em OWL 2

Numa expressão, especificamos uma união anónima de classes, neste caso, das classes *:Man* e *:Woman*. Se de seguida, relacionarmos essa expressão com uma das nossas classes, criamos um axioma, como podemos observar na listagem 2.4.

No capítulo 1, fizemos uma introdução à linguagem OWL 2 [11] como sendo uma linguagem mais expressiva que o RDF, mas em que apenas estender o RDF leva a um raciocínio ineficiente, pois é uma linguagem com um enorme poder expressivo e por isso torna-se indecidível, como referido em [2].

Por essa razão, o OWL 2 foi dividido em duas sub linguagens, cada uma com uma semântica subjacente de forma a irem ao encontro de diferentes requisitos. Essas sub linguagens são o OWL 2 Full e a OWL 2 DL: Direct Semantics, sendo que o OWL 2 Full usa todas as primitivas do OWL 2, enquanto o OWL 2 DL: Direct Semantics foi mapeada para uma lógica de descrição de forma a obter eficiência computacional, como será explicado de seguida, onde iremos falar das vantagens e desvantagens de ambas as sub linguagens.

2.3.1 OWL 2 Full

Esta vertente [29], usa todas as primitivas da linguagem OWL 2, permitindo também a combinação arbitrária destas primitivas com o RDF e o RDFS. Isto permite que seja alterado o significado de primitivas predefinidas ao aplicar as primitivas da linguagem entre si. Por exemplo, no OWL 2 Full, é possível impor uma restrição de cardinalidades na classe de todas as classes, que limita o número de classes que podem ser descritas na ontologia.

A vantagem desta vertente é que é mapeada para *RDF-based semantics*, e como tal é semanticamente e estruturalmente compatível com o RDF, em que qualquer documento OWL 2 Full é também um documento RDF e vice-versa.

A desvantagem desta vertente é que a linguagem é tão poderosa que se tornou indecidível.

2.3.2 OWL 2 DL: Direct Semantics

De forma a obter eficiência computacional, a sub-linguagem OWL 2 DL [25] é mapeada para uma lógica de descrição, nomeadamente a lógica de descrição $SROIQ$. As lógicas de descrição são um sub-conjunto de lógicas de primeira ordem onde o raciocínio eficiente é possível.

Como a lógica de descrição $SROIQ$ subjacente à norma W3C OWL 2 é bastante geral e altamente expressiva, visto que admite diferentes construtores, raciocinar com esta lógica de descrição torna-se bastante complexo, e por isso, perfis como OWL 2 EL, QL e RL foram definidos, para os quais o raciocínio é tratável.

A OWL 2 DL restringe a forma como as primitivas do OWL 2, RDF e RDFS são usadas.

- A linguagem OWL 2 DL não permite a aplicação de primitivas da OWL 2 entre si.
- A linguagem OWL 2 DL apenas permite definir classes de recursos não literais. Todas as classes de OWL 2 DL são instancias de *owl:Class* em vez de *rdfs:Class*.
- OWL 2 DL apenas separa propriedades para as quais o intervalo (*range*) inclua recursos não literais daqueles que se relacionam com valores literais. Todas as propriedades OWL 2 DL são instâncias ou de *owl:ObjectProperty* ou *owl:DatatypeProperty* mas não de ambas.
- Na Linguagem OWL 2 DL, um recurso não pode ser uma classe, propriedade ou uma instância em simultâneo. Os recursos podem partilhar o mesmo nome mas serão sempre tratados de forma diferente pela lógica subjacente.

A vantagem desta limitação na expressividade é que desta forma é possível obter um raciocínio eficiente, mas à custa da perda de compatibilidade com documentos RDF. A linguagem OWL 2 DL faz uso de uma vasta gama de raciocinadores como o Pellet [30], FaCT [32] , RACER [9], e o HermiT [7].

Subjacente ao OWL 2 DL existem vários perfis [26], como o perfil OWL 2 EL, QL e RL, que são subconjuntos de especificações do OWL 2 DL onde alguns são mais expressivos que outros mas nenhum contém a semântica completa do OWL 2 Full.

A motivação para a criação destes perfis é o facto de muitas ontologias existentes tenderem a usar apenas um subconjunto particular dos construtores da linguagem presentes nas lógicas de descrição. Utilizando um poder expressivo mais reduzido é possível aumentar significativamente a eficiência do raciocínio, aumentando a velocidade de resposta.

2.4 Lógicas de descrição

Normalmente, as lógicas de descrição são fragmentos decidíveis da lógica de primeira ordem, definida sobre conjuntos disjuntos finitos e contáveis de conceitos N_c, relações N_r, e indivíduos N_i, correspondendo a predicados unários, predicados binários e constantes, respetivamente. Em cima disto, conceitos complexos são introduzidos baseados em construtores lógicos, que as lógicas de descrição admitem ser usadas.

Uma ontologia O é um conjunto finito de axiomas de inclusão com a forma $C \sqsubseteq D$, onde C e D são ambos conceitos (complexos) ou relações e asserções na forma $C(a)$ ou $R(a,b)$ para conceitos C, relações R, e indivíduos a, b, como referido em [21].

A semântica de tais ontologias é definida por uma *interpretação* $I = (\triangle^I, \cdot^I)$ que consiste num domínio não vazio $\triangle^I$ e numa interpretação $\cdot^I$ por padrão para a lógica de primeira ordem. As tarefas típicas de raciocínio são verificações de modelo, consistência ou classificação o que requer avaliar todas as inclusões de conceitos entre os conceitos atómicos implicados por O, como referido em [21].

A abordagem KRR *(Knowledge Representation and Reasoning)* expressiva mais usada na Web Semântica é baseada em lógicas de descrição , em particular, a linguagem de ontologia OWL é baseada na lógica de descrição $SROIQ(D)$, que é uma norma recomendada pelo W3C *(World Wide Web Consortium)* para a modelação de bases de conhecimento na web semântica.

Como a lógica de descrição $SROIQ$ subjacente à norma W3C OWL 2 é bastante geral e altamente expressiva, visto que admite diferentes construtores, raciocinar com esta lógica de descrição torna-se bastante complexo, e por isso, perfis como OWL 2 EL, OWL 2 QL e OWL 2 RL foram definidos, para os quais o raciocínio é tratável.

- **OWL 2 EL** O perfil EL é uma extensão da lógica de descrição EL. A sua maior vantagem está na capacidade de raciocinar em tempo polinomial sobre ontologias com um grande número de axiomas de classes, e foi concebida para lidar com o poder expressivo de várias ontologias de larga escala na área da saúde, como por exemplo o SNOMED-CT [1], Gene Ontology [2], and GALEN [3].

  Outra característica deste perfil é conseguir lidar com conjunções e com restrições existenciais. Uma das suas diferenças para o OWL 2 DL é que este perfil não contém a restrição *@powl:allValuesFrom*, embora contenha restrições *@prdfs:range* sobre propriedades, que tem um resultado semelhante.

- **OWL 2 QL** Raciocinadores desenvolvidos tanto para OWL 2 DL como para OWL 2 EL são otimizados para raciocinar sobre axiomas de classes, e são normalmente ineficientes em lidar com ontologias que sejam menos complexas em definições de classes mas que contenham um grande número de asserções.

  O perfil QL foi desenvolvido para lidar com pesquisas sobre tais ontologias de forma eficiente, adotando tecnologia de gestão de bases de dados relacionais.

- **OWL 2 RL** O perfil RL é baseado em programas de lógica de descrição [8] e permite a interação entre lógicas de descrição e regras, sendo este perfil o maior fragmento sintático do OWL 2 DL que é implementável usando regras.

  Esta característica é bastante importante, visto que as regras podem ser executadas com eficiência em paralelo, permitindo implementações escaláveis.

  O OWL 2 RL difere dos perfis EL e QL no sentido em que permite uma ligação entre a perspetiva DL e o OWL FULL. Os raciocinadores de regras podem facilmente

[1] http://www.ihtsdo.org/snomed-ct/
[2] http://www.geneontology.org/
[3] http://www.opengalen.org/

ignorar as restrições do OWL DL, por isso, implementações de regras sobre o perfil RL permitem implementar subconjuntos do OWL Full.

2.5 Regras

Outro sub-conjunto de lógicas de primeira ordem com sistemas eficientes de prova (*proof systems*) são os sistemas de regras, neste caso monotónicas, (também conhecidos como lógica de Horn ou programas de lógica definidos), que normalmente aplicam o formalismo de mundo fechado (CWA), ou seja, assumem que todas as expressões que não possam ser provadas sejam falsas.

As regras têm o seguinte formato,

$$A_1, ...A_n \rightarrow B$$

onde A_i e B são fórmulas atómicas que significa que se $A_1, ...A_n$ forem verdadeiros então B também é verdadeiro.

É importante reparar que as lógicas de descrição e os programas de lógica são ortogonais no sentido em que nenhuma é um subconjunto da outra.

Por exemplo, nas lógicas de descrição é impossível definir uma classe de maridos felizes constituído por aqueles que casaram com a sua melhor amiga. Mas este conhecimento é facilmente descrito usando regras, como referido em [2].

$$married(X, Y), bestFriend(X, Y) \rightarrow happySpouse(X)$$

As regras, normalmente, não conseguem afirmar negações/complemento de classes, disjunções/uniões de informação, por exemplo, que uma pessoa ou é um homem ou é uma mulher, ou quantificação existencial, por exemplo, que todas as pessoas têm um pai. Por outro lado, o OWL tem a capacidade de expressar a união de classes e certas formas de quantificação existencial.

Dito isto, vamos analisar outro tipo de regras. Por exemplo, uma loja que dá um *voucher* no caso de ser o aniversário do cliente, pode ser facilmente descrito usando as seguintes regras, também referido em [2].

R1: Se for o aniversário do cliente, então dar um *voucher* ao cliente

R2: Se não for o aniversário do cliente, então não dar um *voucher* ao cliente

Esta solução funciona bem no caso do aniversário do cliente ser conhecido, mas no caso do cliente não fornecer a sua data de nascimento estas regras já não podem ser aplicadas. Portanto, teríamos de alterar estas regras para algo como:

R1: Se for o aniversário do cliente, então dar um *voucher* ao cliente

R2': Se o aniversário não for conhecido, então não dar um *voucher* ao cliente

Contudo a premissa da regra R2' está fora do poder expressivo da lógica de primeira ordem. A solução com as regras R1 e R2 apenas funciona para os casos onde a informação está totalmente disponível.

A lógica de primeira ordem é monotónica no sentido em que, se podemos chegar a uma conclusão, essa conclusão permanece válida mesmo que seja definida nova informação. Mas se a regras R2' é aplicada para "não dar um *voucher*", então a conclusão irá permanecer inválida mesmo que o aniversário do cliente seja posteriormente definido e coincida com a data de compra. Este é tipicamente um comportamento não-monotónico porque a adição de nova informação leva a uma perda da consequência.

De seguida vamos analisar mais detalhadamente a diferença entre as regras monotónicas e as regras não-monotónicas.

2.5.1 Regras monotónicas

Anteriormente, foram apresentados exemplos de regras monotónicas e de regras nã-monotónicas. Nas regras monotónicas, podemos concluir algo apenas com as premissas presentes nas regras sem termos de avaliar outras regras. Outra característica das regras monotónicas é o facto de qualquer nova regra que seja introduzida não alterar as conclusões a partir das regras já existentes.

Analisando ainda o exemplo anterior com as regras R1 e R2, podemos considerar que a lógica de predicados e os seus casos especiais são monotónicos no sentido em que, se for possível chegar a uma conclusão, a informação permanece válida mesmo quando nova informação passa a ser conhecida, como já referido.

2.5.2 Regras não-monotónicas

Até agora sempre que as premissas de uma regra eram provadas, a mesma podia ser aplicada, chegando a uma conclusão. Toda a informação que era adicionada apenas incrementava o conhecimento nas bases de conhecimento, sem nunca por em causa o conhecimento já adquirido.

No sistema de regras não-monotónicas mesmo que todas as premissas sejam conhecidas, a regra pode não ser aplicada, pois é necessário considerar as cadeias de raciocínio que possam contrariar a regra.

Uma regra não-monotónica tem a seguinte forma:

$$H \leftarrow A_1, ..., A_n, notB_1, ..., notB_m$$

Onde a cabeça da regra, H, e todos os A_i com $1 \leq i \leq$ n e B_j com $1 \leq j \leq$ m no corpo de r são átomos, como mencionado em [21]. A introdução da negação (*not*), torna esta variante não-monotónica.

Esta variante das regras, é a utilizada nas bases de conhecimento MKNF híbridas onde iremos analisar em detalhe de seguida.

2.6 Bases de conhecimento MKNF híbridas

Hybrid MKNF (minimal knowledge and negation as failure) knowledge bases, como referidas em [24], são essencialmente fórmulas de MKNF restringidas de certa forma. A lógica do MKNF estende a lógica de primeira ordem com dois operadores modais, K e *not*. As bases de conhecimento híbridas *MKNF* consistem em dois componentes: uma *description logic knowledge base* decidível e traduzível para lógica de primeira ordem e um conjunto finito de regras de operadores modais.

Foram definidas duas versões diferentes de semânticas ([19] [24]), embora neste caso, vamos focar-nos apenas na *Well-founded* [19], devido à sua eficiência do ponto de vista computacional e por usar a estratégia de *top-down querying* que não necessita de avaliar todo o modelo, como iremos mencionar mais à frente.

As *MKNF knowledge bases* presentes em [1] combinam uma ontologia e um conjunto de regras não-monotónicas. Uma regra r tem a forma de $H \leftarrow A_1,...,A_n,notB_1,...,notB_m$, onde a cabeça da regra, H, e todos os A_i com $1 \leq i \leq$ n e B_j com $1 \leq j \leq$ m no corpo de r são átomos. Um programa P é constituído por um conjunto finito de regras, O é uma ontologia, e as bases de conhecimento MKNF K são um par (O,P). Uma regra é válida se todas as suas variáveis ocorrerem pelo menos uma vez em A_i com $1 \leq i \leq$ n, sendo K válido se todas as suas regras forem válidas.

Entre as várias propostas que combinam regras e ontologias, *Hybrid MKNF* [19] é a única que satisfaz todos os seguintes critérios.

- Faithfulness: A integração de DLs com regras deve preservar a semântica de ambos os formalismos - isto é, a semântica de uma base de conhecimento híbrida em que um componente é vazio, deve ser o mesmo que a semântica de outro componente. Por outras palavras, a adição de regras a uma DL não deve alterar a semântica da DL e vice-versa.

- Tightness: As regras não devem ser colocadas numa camadas acima de uma DL ou vice-versa. Em vez disso, a integração entre uma DL e as regras devem estar relacionadas no sentido de que tanto as DL como o componente das regras devem poder contribuir para as consequências um do outro.

- Flexibility: O formalismo híbrido deve ser flexível e permitir que se veja o mesmo predicado sobre as interpretações de mundo aberto e fechado. Isto permite enriquecer uma DL com consequências não-monotónicas das regras, e enriquecer as regras com as capacidades de raciocínio ontológico descrito por uma DL.

- Decidability: Para obter um formalismo útil que possa ser usado em aplicações como a Web Semântica, o formalismo híbrido deve ser pelo menos decidível e, de preferência, da complexidade do pior cenário.

2.7 Protégé

O *Protégé* [4] é uma ferramenta *open-source* bastante completa de edição de ontologias com total suporte para a linguagem ontológica OWL 2, fazendo ligações diretas em memória para raciocinadores lógicos de descrição como HermiT [7] e Pellet [30]. Sendo uma ferramenta bastante versátil, o *Protégé* suporta a criação ou edição de uma ou mais ontologias numa única *workspace* utilizando uma interface completamente customizável. O *Protégé* possuí também ferramentas de visualização que permitem uma navegação interativa pela ontologia.

O *Protégé* suporta também operações de *refactor*, incluindo a fusão de ontologias, assim como mover axiomas entre ontologias, renomear múltiplas entidades, entre outras funcionalidades.

Uma das grandes características do *Protégé* é permitir a instalação *"plugins"*. Para isso, deve ser gerado um ficheiro do tipo *"jar"* do plugin que se está a desenvolver e adicioná-lo à pasta *"plugins"* localizada na *root* da instalação do *Protégé*.

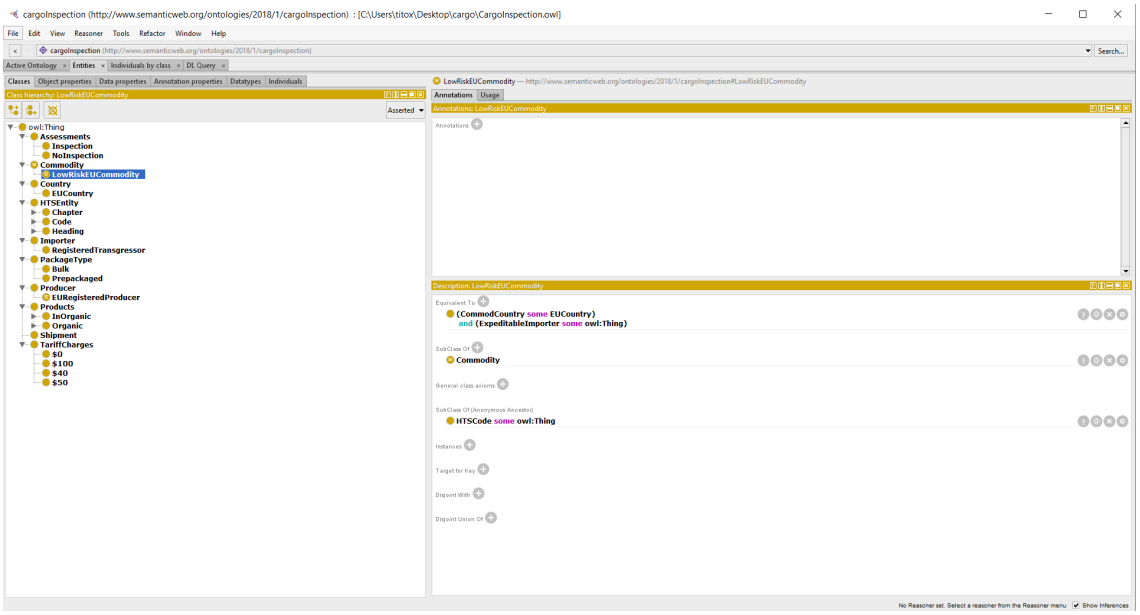

Figura 2.1: Interface do *Protégé* com uma ontologia carregada

A figura 2.1 mostra a versão simples do *Protégé*, ou seja, sem qualquer *plug-in* instalado, onde é possível ver parte de uma hierarquia de uma ontologia carregada no lado esquerdo da interface e no lado direito podemos ver a informação sobre a classe que

[4] https://protege.stanford.edu

temos selecionada, como, as suas classes equivalentes, subclasses, instâncias, disjunções, entre outros.

O *Protégé* permite ainda a execução de perguntas em *SPARQL*[5], que executa perguntas sobre a linguagem RDF, em *SWRL*[6], que combina o OWL DL com um subconjunto da *Rule Markup Language*, e a *DL Query* que permite fazer pesquisas sobre a ontologia, em que esta deve ser previamente classificada por um raciocinador antes de permitir fazer pesquisas.

2.8 NoHR plug-in

O NoHR é um *plug-in* para o editor de ontologias *Protégé* que permite que os utilizadores executem perguntas sobre bases de conhecimento compostas tanto por ontologias OWL 2 EL, QL ou RL e até combinação entre operadores. Nesse caso é utilizado o Hermit e o Konclude para fazer a combinação das características destes perfis, e um conjunto de regras não-monotónicas de raciocínio, sendo o NoHR o primeiro raciocinador híbrido para o editor *Protégé*.

A utilização destes perfis pelo NoHR depende concretamente da ontologia. Uma condição determina quando a ontologia deve entrar nos perfis ou não, permitindo também ao utilizador definir manualmente um dos perfis.

Usando uma estratégia de raciocínio *top-down*, que significa que apenas a parte da ontologia e das regras que é relevante para a pesquisa é analisada, o NoHR combina a capacidade do ELK [17] (no caso do OWL 2 EL) e uma tradução direta dedicada para o OWL QL e OWL RL, respetivamente, mas também o Konclude [31] e o HermiT [7], respetivamente, com o *rule engine XSB Prolog* de forma a produzir tempos de resposta interativos bastante eficientes.

O *input* para o *plug-in* consiste num ficheiro OWL escrito numa das lógicas de descrição descritas na secção 2.4, que pode ser lido e editado no *Protégé*, e num ficheiro de regras. Para este último, o NoHR contem uma aba específica, chamada *NoHR Rules*, que permite que o utilizador carregue um ficheiro de regras onde poderá editar o conjunto de regras.

O NoHR fornece também outra aba onde permite visualizar as regras, mas o objetivo principal desta aba é providenciar uma interface para pesquisas nas bases de conhecimento combinadas, como mencionado em [21].

As figuras 2.2 e 2.3 mostram as abas do *plug-in* NoHR , respetivamente a aba de regras e a aba de *queries*. Na aba de regras é possível ver e editar o conjunto de regras não-monotónicas existentes no programa, é também a partir desta aba que se carregam os ficheiros de regras do NoHR. Na aba de *queries* é possível fazer perguntas (na secção direita da interface da figura 2.3), onde as respostas serão apresentadas em baixo, no formato de uma tabela.

[5] SPARQL Protocol and RDF Query Language
[6] Semantic Web Rule Language

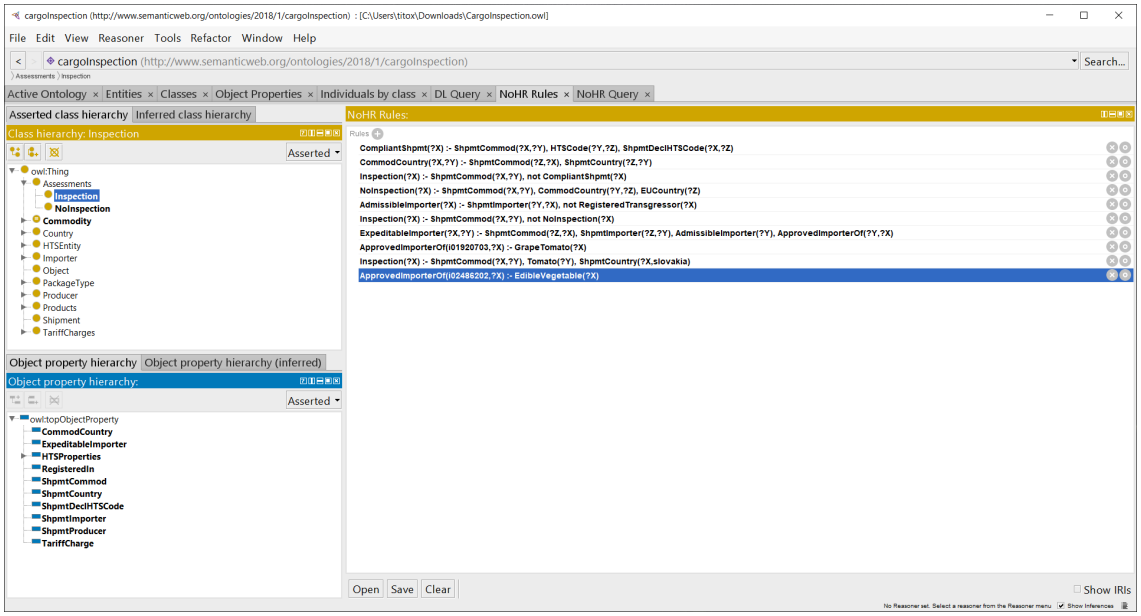

Figura 2.2: Interface do *Protégé* na aba das regras do NoHR

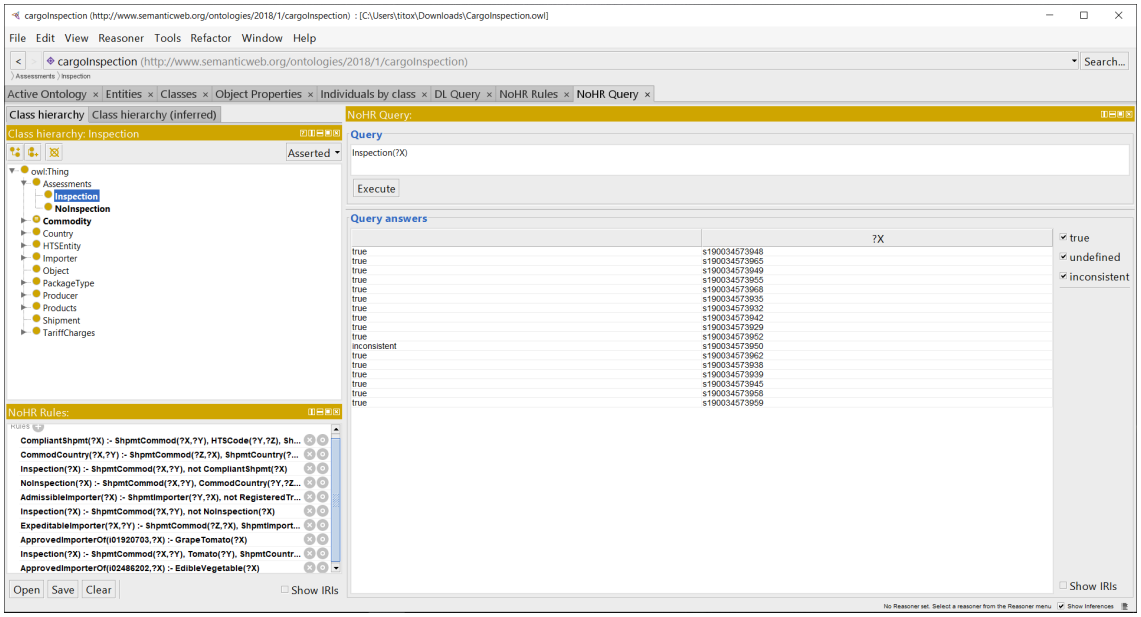

Figura 2.3: Interface do *Protégé* na aba da *queries* do NoHR

Quando é feita uma pesquisa, o tradutor é iniciado, utilizando um dos raciocinadores integrados da ontologia OWL, como o ELK, HermiT e o Konclude que transformam os axiomas de ontologia num conjunto de regras que são equivalentes aos axiomas de ontologia no que diz respeito a responder a *ground queries*. O resultado da tradução é combinado com as regras não-monotónicas dadas em P, que serão transformadas mais à frente se for necessário detetar inconsistências.

O resultado é então usado como *input* para a estratégia de pesquisa *top-down* no XSB Prolog que utiliza a *well-founded semantics* para programas em lógica [5]. A transferência

para o XSB é realizada via InterProlog [7], que é uma ferramenta *front-end open-source* em Java que permite a comunicação entre o Java e o Prolog.

De seguida, a pesquisa é enviada via InterProlog para o XSB, e as respostas são retornadas para o processador de pesquisas, que com todas as respostas cria uma tabela em que mostra para que substituição de variáveis é possível obter verdade, indefinido, ou avaliações inconsistentes (ou apenas mostra o valor de verdade para a *ground query*). O XSB por si só não responde às perguntas de forma eficiente com uma estratégia *top-down*, usando o *tabling*, também previne ciclos infinitos, o que assegura o fim do processo de pesquisa.

Quando a pesquisa termina, o utilizador pode fazer outras pesquisas, que serão simplesmente enviadas pelo sistema sem que seja necessário qualquer tipo de pré-processamento repetido. Se o utilizador fizer alterações na ontologia ou nas regras, o sistema recompila, mas apenas na parte que realmente foi alterada pelo utilizador.

2.8.1 Mapeamentos para bases de dados

A última versão do NoHR adiciona uma terceira componente ao NoHR , denominada *"Extended Hybrid Knowledge Base"*, que faz a integração com bases de dados através de mapeamentos, permitindo que os utilizadores carreguem as ontologias e as regras diretamente de bases de dados, sem a necessidade de extrair os dois ficheiros.

Como hoje em dia a maior parte da informação encontra-se armazenada em bases de dados, é necessário que o utilizador tenha acesso a essa informação de forma direta, embora na versão anterior do NoHR isso não fosse possível. O utilizador teria de extrair a informação da base de dados para ficheiros para poder carregar no NoHR através das interfaces. A última versão inclui a funcionalidade que permite fazer ligações a bases de dados através da já referida terceira componente. A nova base de conhecimento é composta pela ontologia, por um conjunto de regras não-monotónicas e por um número variável de bases de dados.

Os mapeamentos de bases de dados são implementados usando *drivers* ODBC, que permitem que o raciocinador suporte a integração com os maiores sistemas de bases de dados, como referido em [16].

Quando a primeira pergunta é executada, os mapeamentos de bases de dados são traduzidos para regras e integrados com o conjunto de regras não-monotónicas e com a representação de regras da ontologia. Posteriormente, todo o conjunto é usado como *input* para o XSB.

Esta solução permite a poupança de tempo e reduz o consumo de memória durante o processamento evitando o armazenamento de informação irrelevante na memória, recorrendo a informação armazenada em bases de dados externas, ao utilizar apenas as instâncias relevantes no processo de raciocínio.

[7] http://interprolog.com/java-bridge/

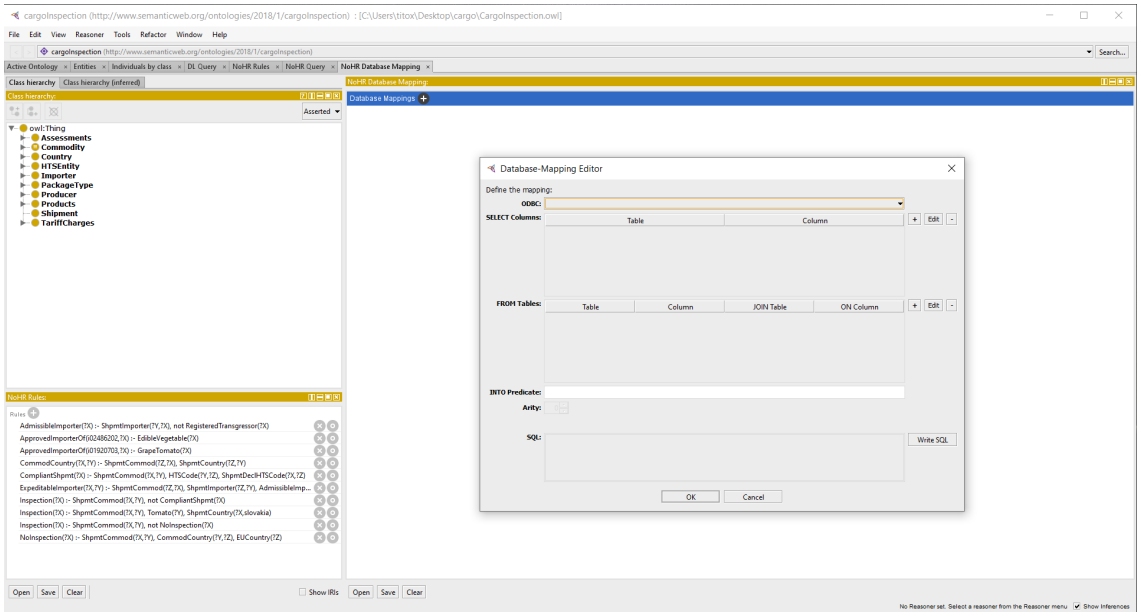

Figura 2.4: Interface do *Protégé* na aba de mapeamentos para bases de dados do NoHR com o formulário de criação de mapeamentos

Na figura 2.4, é possível observar o formulário de criação de um mapeamento para bases de dados no NoHR.

2.8.2 Perfis OWL 2 Suportados

O NoHR suporta o uso de ontologias escritas em qualquer um dos perfis de OWL 2 e até combinar características de cada um deles. Neste caso, o raciocínio das lógicas de descrição não é polinomial, mas com o uso de raciocinadores eficientes de lógicas de descrição , como o HermiT e o Konclude, é possível compensar esta questão na prática apesar da alta complexidade do pior caso.

Como referido na secção 2.4, o uso dos perfis depende concretamente da ontologia, em que, uma condição determina que perfil será utilizado pelo NoHR para o raciocínio.

Para os perfis OWL 2, foram desenvolvidos módulos de tradução específicos. Para o $EL^{+}_{\bot}$, que é a lógica de descrição subjacente ao perfil OWL 2 EL, o raciocinador ontológico ELK, à medida do $EL^{+}_{\bot}$ e consideravelmente mais rápido que outros raciocinadores quando comparado o tempo de classificação, em que, é usado para classificar a ontologia resultante ao normalizar uma ontologia O para assegurar que não são perdidas quaisquer derivações durante a tradução em regras.

Para o $DL-Lite_R$, que é a lógica de descrição subjacente ao perfil OWL 2 QL, é usada uma tradução direta e dedicada sem a necessidade de uma classificação prévia, introduzindo, como alternativa, alguns predicados auxiliares para compensar a falta de axiomas inferidos (ver [15] e [3] respetivamente, para mais detalhes sobre as duas abordagens).

O módulo DLP, que é a lógica de descrição subjacente ao perfil OWL 2 RL, como este perfil suporta a tradução direta em regras por defeito, esta abordagem faz uso de uma

tradução direta que não requer quaisquer predicados auxiliares, como mencionado em [21].

Com a preocupação de que uma ontologia possa não entrar em nenhum perfil, um novo módulo geral de tradução é usado e segue uma metodologia usada no $EL^{+}_{\perp}$, isto é, normalização (parcial), classificação, e tradução dos axiomas inferidos, onde a função de tradução por si só resulta numa junção destes para perfis únicos.

A razão pela qual se utilizam dois raciocinadores gerais de lógicas de descrição é a seguinte. No caso do Konclude, foi demonstrado que é o raciocinador mais eficiente até ao momento [31], mas infelizmente, ao contrário do *Protégé* não foi desenvolvido em Java, mas sim em C++, e por isso é necessário um esforço extraordinário que retira eficiência ao raciocínio, porque não existe uma interface nativa de Java compatível com a versão atual do OWL API. No caso do HermiT, não sofre deste problema, mas tem certas limitações quando raciocina com grandes ontologias [7]. De qualquer forma, ambos os raciocinadores estão integrados e o utilizador pode escolher qual dos dois usar.

2.9 WebProtégé

O *WebProtégé* [8] é a versão web da aplicação *Protégé* de uso livre, desenvolvido sobre a versão desktop *Protégé*, onde o código fonte pode ser descarregado a partir do *github*, permitindo que qualquer pessoa possa utilizar e contribuir assim para o desenvolvimento da aplicação.

O utilizador interage com o cliente da aplicação que corre em JavaScript em qualquer navegador de Internet. A interface do *WebProtégé* está implementada com uma estrutura de Abas que contêm janelas que são totalmente personalizáveis pelo utilizador. O servidor do *WebProtégé* corre num *servlet container*, como por exemplo o Tomcat[9]. Os *servlet container* permitem que o servidor envie páginas para o cliente de forma dinâmica, e no caso do *WebProtégé* o servidor é desenvolvido em Java, como mencionado em [33].

Esta versão também é totalmente compatível com a linguagem de ontologia OWL 2 assim como o *Protégé*, mas como o *WebProtégé* é uma aplicação web retira do utilizador toda a responsabilidade de instalação e configuração local da aplicação. Desta forma torna-se mais simples criar, atualizar e editar ontologias, sendo ainda possível partilhá-las com outros utilizadores, permitindo a colaboração de uma equipa de uma forma mais dinâmica num projeto. Embora o *WebProtégé* tenha as funcionalidades do *Protégé*, na interação com projetos, não permite a instalação de *plugins* por parte de terceiros de forma direta como acontece no *Protégé*, perdendo desta forma muito conteúdo criado por terceiros existente no *Protégé*.

Atualmente existe um servidor público para a utilização livre do *WebProtégé*. Para utilizar o *WebProtégé* , o utilizador apenas necessita criar uma conta, sem ser necessário efetuar alguma instalação ou configuração, o que torna bastante apelativo a sua utilização.

[8] https://webprotege.stanford.edu
[9] O Tomcat é uma implementação de uso livre de um *Java servlet*

O utilizador pode carregar uma ontologia para o *WebProtégé*, utilizando um ficheiro com a mesma estrutura que usa para o *Protégé*. Mas como esta versão não faz a integração de ontologias com as regras não-monotónicas, as funcionalidades que os utilizadores podiam tirar do NoHR utilizando o *Protégé* não são possíveis utilizando o *WebProtégé*.

Outra vantagem do *WebProtégé* em relação ao *Protégé*, é que as alterações do utilizador no *WebProtégé* ficam imediatamente guardadas no servidor, sem ser necessário guardar a ontologia, as regras não-monotónicas e os mapeamentos de bases de dados em ficheiros. Para guardar a informação sobre os projetos, utilizadores e a restante informação relacionada com o servidor, o *WebProtégé* utiliza o *MongoDB* que é uma base de dados não relacional.

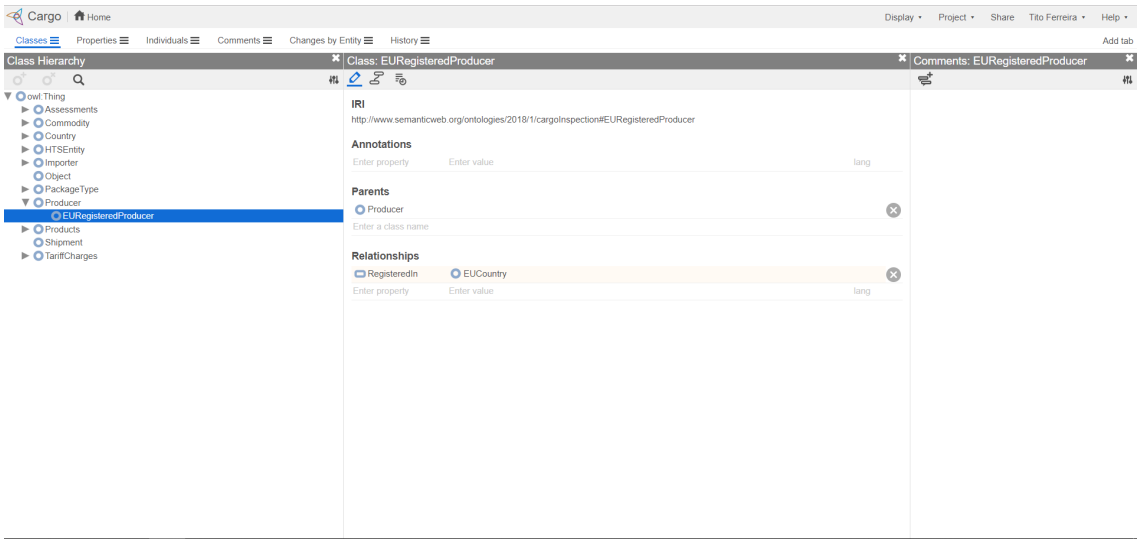

Figura 2.5: Interface do *WebProtégé* com uma ontologia carregada

A figura 2.5 mostra a interface do *WebProtégé* com uma ontologia carregada. Como podemos observar na figura 2.5, a estrutura da interface inicial de um projeto do *WebProtégé* é semelhante ao *Protégé* sem *plugins* instalados, onde é possível observar a hierarquia de classes da ontologia carregada.

O *WebProtégé* foi desenvolvido usando o Google Web Toolkit que é uma ferramenta *open-source*, licenciada sobre o *Apache License 2.0*, de desenvolvimento de aplicações web. É adequada para aplicações complexas como o *WebProtégé*, visto que faz a otimização do código do utilizador, utilizando técnicas como o *in-lining method* que reduz o impacto na performance na chamada de métodos, remoção de código não utilizado e otimização de strings, entre outras otimizações, maximizando desta forma o desempenho da aplicação.

O GWT SDK fornece um conjunto base de Java APIs e de *Widgets*, isto permite criar aplicações AJAX[10] *front-end* na linguagem de programação Java, sendo posteriormente compilado pelo GWT em código Javascript otimizado e pronto a correr em todos os *browsers*.

[10] Asynchronous JavaScript and XML, que é um conjunto de técnicas de desenvolvimento web *front-end* que permite criar aplicações assíncronas

O GWT assegura também as comunicações entre o cliente e o servidor, quer sejam no formato JSON, XML ou no *Remote Procedure Call* (RPC) otimizado do GWT, sendo este último o que é usado no *WebProtégé*. O GWT permite também que o servidor seja escrito em diferentes linguagens de programação, sendo que no *WebProtégé* é utilizado também a linguagem Java no lado do servidor.

2.10 Introdução ao MongoDB

O *Protégé* não guarda qualquer tipo de informação que o utilizador insira no *plug-in* entre diferentes inicializações do *Protégé*, no caso das regras não-monotónicas e dos mapeamentos para bases de dados. Caso o utilizador pretenda guardar a informação entre várias sessões do *Protégé*, é necessário gravar a informação para ficheiros e carregar a informação pretendida novamente para o programa.

No *Protégé*, sempre que um utilizador pretende trabalhar num projeto, tem necessariamente de carregar os ficheiros da ontologia, de regras e de mapeamentos para as bases de dados para o programa, o que torna este processo de inicialização repetitivo. Em relação às definições do NoHR, estas são guardadas pelo *Protégé* e desta forma permanecem armazenadas entre diferentes inicializações do programa.

No *WebProtégé* a maior parte da informação do servidor, como as contas dos utilizadores e os seus acessos, informação sobre os seus projetos e as definições dos projetos são armazenadas na base de dados *MongoDB*.

O *MongoDB* é uma base de dados não relacional orientada para a utilização em aplicações web, visto que permite uma grande escalabilidade. Usando a estratégia *Document Oriented Storage*, em que a informação é armazenada no formato de um documento JSON tornando o tratamento da informação simplificado para as aplicações.

A estrutura do *MongoDB* é um pouco diferente da estrutura das bases de dados relacionais. No *MongoDB* não é necessário definir tabelas em que todos os tuplos na tabela têm de ter exatamente a mesma estrutura, como acontece nas bases de dados relacionais. No *MongoDB* existem as *collections* que permitem armazenar os diferentes documentos, em que os documentos podem ter uma estrutura diferente dentro da mesma *collection*.

RDBMS	**MongoDB**
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default keyID provided by mongodb)

Tabela 2.1: Diferenças estruturais entre bases de dados relacionais e o *MongoDB*.

```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

Listagem 2.5: Estrutura de um documento no MongoDB

```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
               death: new Date('Jun 07, 1954'),
               contribs: [ "Turing machine", "Turing test", "Turingery" ],
               views : NumberLong(1250000)
             }
```

Listagem 2.6: Exemplo prático de um documento no MongoDB

Na tabela 2.1, são apresentadas as diferenças conceptuais entre as bases de dados relacionais e o *MongoDB*, que é uma base de dados não-relacional.

Dentro de cada *collection* podemos criar documentos com a estrutura pretendida. Um documento é um registo no formato BSON, que é uma representação binária de um documento JSON. Para diferenciar os vários documentos de uma *collection*, o *MongoDB* atribui internamente um identificador único a cada documento.

Nas listagens 2.5 e 2.6, é possível observar a estrutura de um documento do *MongoDB* no formato JSON e um exemplo prático, respetivamente, em que os valores no primeiro exemplo podem ser do tipo *String*, *Boolean*, *Double*, *Array*, entre muitos outros e até mesmo outros documentos embebidos.

2.11 Integração com o Apache Maven

Tanto o *WebProtégé* como o NoHR têm a estrutura do projeto organizada usando o Apache Maven que é um software de gestão de projetos. É baseado no conceito de um *Project Object Model* (POM), em que desta forma é possível definir a estrutura de um projeto através de um ficheiro XML, que contém dependências para outros módulos e componentes, os plugins necessários, os diretórios e a ordem da construção do projeto. Automatiza tarefas de compilação, empacotamento, e instalação de projetos, e permite fazer download dinamicamente de bibliotecas públicas Java e de plugins Maven a partir de um repositório central, guardando-as numa *cache* local.

De seguida, vamos analisar os módulos tanto do *WebProtégé* como do NoHR no Maven, onde vamos analisar a sua estrutura e explicar as funções inerentes a cada um dos

módulos.

2.11.1 Estrutura do WebProtégé

A aplicação *WebProtégé* está estruturada num projeto que contém um conjunto de módulos organizados através do Maven, dentro dos vários módulos destacam-se os três principais que são, o cliente, o servidor e *shared* que contêm classes usadas tanto no cliente como no servidor, existindo também três módulos usados como auxiliares destes três módulos principais.

- **Módulo Client** Este módulo contém toda a informação e lógica das interfaces da aplicação. Todo o código neste módulo é compilado para JavaScript no momento da compilação, desta forma é necessário ter em atenção que nem todo o código Java é convertível para JavaScript, e por isso existem algumas restrições de bibliotecas Java que podem ser utilizadas neste módulo.

- **Módulo Server** Este módulo é o ponto de acesso principal do servidor e contém também toda a informação necessária ao funcionamento da aplicação. É neste módulo também que o código front-end já compilado fica armazenado, as estratégias de codificação e descodificação dos objetos Java que são trocados nos pedidos, estratégias de *cache* por parte do servidor entre outras. No caso do *WebProtégé* o servidor é desenvolvido usando a linguagem Java, tendo este módulo liberdade total na utilização de bibliotecas Java, contrariamente ao módulo cliente que é convertido para JavaScript no momento da compilação.

- **Módulo Shared** Neste módulo estão classes usadas tanto no módulo cliente como no módulo servidor, sendo que o código deste módulo também não é convertido para JavaScript, embora caso as classes sejam usadas em pedidos do cliente para o servidor, essas classes têm de ser serializáveis para que os objetos Java sejam traduzidos para bytes, de forma a poderem ser enviadas pela rede entre os vários pedidos do cliente e do servidor.

- **Módulo Cli** Este módulo contém maioritariamente *scripts* de configuração do servidor.

- **Módulo Server-Core** Este módulo contém as classes base do servidor, é neste módulo que é feito o tratamento dos pedidos que chegam ao servidor.

- **Módulo Shared-Core** Este módulo funciona como um auxiliar do módulo Shared, em que é responsável por guardar a lógica de permissões do utilizadores ás funcionalidades da aplicação entre outras funções.

2.11.2 Estrutura do NoHR

O projeto do NoHR também está organizado através do Maven e contém três módulos na sua estrutura.

- **Módulo plugin** Este módulo contém maioritariamente código *front-end* do plugin NoHR que apenas é compatível com o *Protégé*.
- **Módulo reasoner** Este módulo é responsável por toda a lógica do plugin NoHR, como a tradução da ontologia, o processamento das perguntas e da regras não-monotónicas para os raciocinadores, comunicação com o XSB através do interprolog, entre outros.
- **Módulo benchmark** Este módulo contém classes para avaliação de eficiência das respostas às perguntas, tirando as respetivas métricas.

Capítulo 3

Estrutura do servidor e das novas funcionalidades da aplicação

Neste capítulo vamos abordar as estratégias usadas no desenvolvimento do NoHR Web e como foi integrado o raciocinador do NoHR numa estrutura como um servidor, permitindo a utilização simultânea das funcionalidades do NoHR.

3.1 Introdução do NoHR Web

A extensão do NoHR para a web foi desenvolvida sobre a aplicação já existente *WebProtégé*, utilizando a *framework* Google Web Toolkit, que é a ferramenta usada no desenvolvimento do *WebProtégé*, mantendo a estrutura dos módulos já existente no NoHR, criando assim o NoHR Web.

A estrutura do NoHR Web contempla a integração dos módulos Maven do NoHR *plug-in* com os módulos do *WebProtégé* referidos no capítulo 2. No caso do *módulo plugin* do NoHR, a maior parte da informação não foi utilizada pois contém maioritariamente código *front-end* para o *Protégé* que não é de todo compatível com o código *front-end* do *WebProtégé*.

Por esta razão, toda a estrutura *front-end* do NoHR Web foi desenvolvida de raiz, mantendo a estrutura da interface do *WebProtégé* para uma melhor integração funcional e visual da aplicação, visto que, a versão *desktop* do NoHR foi desenvolvida para o *Protégé* em que a *framework* de desenvolvimento de código *front-end* é intrínseco ao Java, o mesmo não acontece no NoHR Web e por isso foi necessário fazer o desenvolvimento da interface utilizando o Google Web Toolkit.

A estrutura da interface do *WebProtégé* é semelhante à do *Protégé*, ambos fazem a gestão das várias janelas de um projeto através de uma estrutura de abas. Por esta razão

as interfaces referentes às janelas do NoHR no NoHR Web são semelhantes às janelas do NoHR no *Protégé*.

Para esta integração, algumas dependências do NoHR tiveram de ser atualizadas para as versões usadas no *WebProtégé* e outras tiveram de ser substituídas por dependências permitidas no *WebProtégé*. Nomeadamente a dependência *owlapi-osgidistribution* foi substituída pela *owlapi-apibinding* da biblioteca owlapi e atualizada da versão 4.2.6 para a 4.5.13, que é a versão utilizada pelo *WebProtégé* da biblioteca owlapi.

No NoHR Web todo o poder de processamento dos raciocinadores estão no servidor, permitindo que as pesquisas dos utilizadores não sejam afetadas pelo dispositivo que estão a usar. Foi criada toda a estrutura no servidor para tratar dos pedidos dos utilizadores e do processo de raciocínio nos diferentes projetos, assim como guardar a informação subjacente ao NoHR no servidor. A informação sobre as regras não-monotónicas e os mapeamentos para as bases de dados são agora armazenados numa base de dados associando-as a um respetivo projeto. Desta forma o utilizador já não precisa de guardar as regras não-monotónicas e os mapeamentos para bases de dados em ficheiros entre as várias utilizações do NoHR.

No NoHR Web toda a informação referente ao NoHR fica guardada juntamente com o projeto, retirando qualquer responsabilidade ao utilizador de guardar a informação para futura utilização.

3.2 Configurações do NoHR Web

Para instalar o NoHR Web numa máquina servidor, existe uma série de passos que têm de ser seguidos.

Quando o projeto é compilado através do Maven, é gerado um ficheiro com a extensão ".war". Este ficheiro contém toda a informação da aplicação web.

É necessário instalar um *sevlet container* na máquina onde pretendemos instalar o servidor, como por exemplo, o Apache Tomcat[1], que é um *software* livre que implementa um *Java Servlet* e será o responsável por hospedar o servidor. No Tomcat é possível configurar os parâmetros desde a memória mínima e máxima que queremos dedicar ao servidor, como localização de ficheiros temporários e ficheiros de log, entre muitas outras informações relevantes para o servidor.

A primeira vez que o gestor do servidor tentar entrar no servidor, é necessário baixar e executar um *Script* para fazer a criação de uma conta de administrador. Através dessa conta fazer várias configurações no servidor, como se autoriza que os utilizadores possam criar contas livremente, se permite o upload de ficheiros de ontologias na criação de projetos, o tamanho máximo que esses ficheiros podem ter, entre outras. A forma de instalação e configuração do NoHR Web é em grande parte semelhante à do *WebProtégé*, e por isso a informação para a configuração do *WebProtégé* é válida para o NoHR Web, e

[1] http://tomcat.apache.org/

é possível encontrá-la no seu guia de instalação [2]. Nesta página, está o tutorial completo de configuração do *WebProtégé* que neste caso também se aplica ao NoHR Web, que passa por toda a configuração que é necessária fazer para lançar o servidor no Tomcat.

Se o gestor do servidor pretender que o *MongoDB* fique alojado na mesma máquina do servidor, terá de efetuar também a instalação e a configuração desta base de dados para a utilização pelo NoHR Web.

3.3 Integração com o MongoDB

O NoHR web foi desenvolvido com o objetivo de fazer uso da característica de armazenamento de informação sobre os projetos intrínseca ao *WebProtégé*, criando desta forma uma aplicação web que integra as funcionalidades do NoHR. Por esta razão, o desenvolvimento utilizando o *MongoDB* ao invés de uma outra base de dados foi devido ao facto do *WebProtégé* já fazer uso desta base de dados.

Para fazer a ligação ao *MongoDB* o gestor do servidor pode configurar a localização da base de dados, por omissão o servidor irá procurar pelo *MongoDB* em *localhost*. Embora esta informação possa ser editada num ficheiro de configurações do servidor. O *MongoDB* não tem de estar obrigatoriamente na mesma máquina do servidor.

Toda a informação relacionada com o NoHR gerada pelo utilizador durante a utilização do NoHR Web fica armazenada no *MongoDB*, sem que o utilizador precise de se lembrar de guardar uma versão do seu trabalho a cada alteração. Desta forma o seu trabalho fica permanentemente guardado no servidor.

Por outro lado, é importante referir que o utilizador nunca comunica diretamente com a base de dados *MongoDB*, sempre que é executado um pedido pelo cliente em que seja necessária informação armazenada no *MongoDB*, o utilizador executa o pedido ao servidor e é o servidor que faz o pedido pela informação, que o utilizador pretende, ao *MongoDB*. O *MongoDB* retorna a informação para o servidor e é o servidor que faz o tratamento da informação e a envia para o cliente.

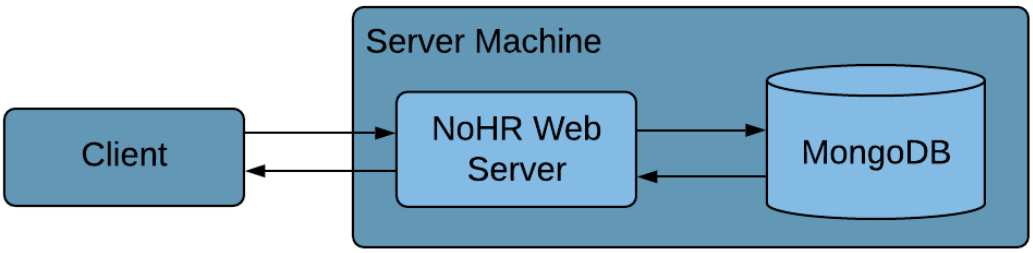

Figura 3.1: Esquema de comunicação entre o cliente, servidor e o *MongoDB*

[2] https://github.com/protegeproject/webprotege/wiki/WebProtégé-4.0.0-beta-x-Installation

Na figura 3.1, está representado, de forma simples, o percurso da comunicação entre um cliente, o servidor e o *MongoDB*, onde é possível observar que apenas o servidor entra em contacto direto com o *MongoDB*, como já referido.

Para armazenar a informação referente ao NoHR, como as regras não-monotónicas, os mapeamentos para bases de dados, e as definições do raciocinador e as definições das ligações a bases de dados através dos mapeamentos foram criadas *collections* no *MongoDB*.

Mais concretamente, foram criadas quatro novas *collections* na estrutura do *WebProtégé* no *MongoDB* que estão enumeradas de seguida.

- **NoHRDatabaseMappings** Guarda os mapeamentos para as bases de dados do NoHR para os projetos.
- **NoHRDatabaseSettings** Guarda as definições dos mapeamentos para as bases de dados do NoHR para os projetos.
- **NoHRRules** Guarda as regras não-monotónicas do NoHR para os projetos.
- **NoHRSettings** Guarda as definições do NoHR para os projetos.

De seguida iremos abordar mais em detalhe estes quatro itens.

3.3.1 Regras não-monotónicas no MongoDB

Como mencionado anteriormente, o utilizador pode criar e editar um conjunto de regras não-monotónicas associado a um respetivo projeto, que são posteriormente armazenadas no *MongoDB*.

Quando o utilizador entra num projeto na aba que contém as regras não-monotónicas, é feito um pedido ao servidor que devolverá ao utilizador as regras do projeto, em que o utilizador está a trabalhar e que estão contidas no *MongoDB*.

Estas regras não-monotónicas ficam armazenadas na *collection* **NoHRRules**, referida na secção 3.3, em que cada documento dentro desta *collection* guarda as regras de um determinado projeto. Cada documento desta *collection* é constituído por um identificador único do documento (fornecido pelo *MongoDB*), um identificador único do projeto e uma lista de regras, ambos no formato de Strings.

Na listagem 3.1, é possível observar as regras não-monotónicas, referentes a um projeto, no formato JSON armazenado no *MongoDB*.

Todas as regras guardadas no *MongoDB*, são previamente avaliadas por um *parser*, no servidor do NoHR, que irá avaliar a sua sintaxe. Caso a regra tenha algum erro sintático, esta não será adicionada ao conjunto de regras na base de dados. Isto garante que todas as regras não-monotónicas presentes no *MongoDB* sejam válidas em termos sintáticos.

Quando uma pergunta é executada, o servidor vai buscar todas as regras não-monotónicas do respetivo projeto ao *MongoDB*, faz a conversão das regras, usando o *parser* do NoHR,

```
{
    "_id":{"$oid":"5e1459b59cdb8238f2f6ea0b"},
    "ProjectID":"ee38efa7-8a69-4846-a81b-2fbeb561497b",
    "Rules":
        [
            "AdmissibleImporter(?X)
                :- ShpmtImporter(?Y,?X), not RegisteredTransgressor(?X)",
            "ApprovedImporterOf(i02486202,?X)
                :- EdibleVegetable(?X)",
            "ApprovedImporterOf(i01920703,?X)
                :- GrapeTomato(?X)",
            "CommodCountry(?X,?Y)
                :- ShpmtCommod(?Z,?X), ShpmtCountry(?Z,?Y)",
            "CompliantShpmt(?X)
                :- ShpmtCommod(?X,?Y), HTSCode(?Y,?Z), ShpmtDeclHTSCode(?X,?Z)",
            "ExpeditableImporter(?X,?Y)
                :- ShpmtCommod(?Z,?X), ShpmtImporter(?Z,?Y),
                AdmissibleImporter(?Y), ApprovedImporterOf(?Y,?X)",
            "Inspection(?X)
                :- ShpmtCommod(?X,?Y), not CompliantShpmt(?X)",
            "Inspection(?X)
                :- ShpmtCommod(?X,?Y), Tomato(?Y), ShpmtCountry(?X,slovakia)",
            "Inspection(?X)
                :- ShpmtCommod(?X,?Y), not NoInspection(?X)",
            "NoInspection(?X)
                :- ShpmtCommod(?X,?Y), CommodCountry(?Y,?Z), EUCountry(?Z)"
        ]
}
```

Listagem 3.1: Exemplo de um documento de regras não-monotónicas do MongoDB

do formato *String* para regras não-monotónicas do NoHR que são constituídas por uma cabeça que é um átomo e um corpo que é um conjunto de literais.

3.3.2 Definições do NoHR no MongoDB

Cada projeto no NoHR Web tem um conjunto de definições do NoHR que têm de ser guardadas de forma a garantir que mantêm as alterações efetuadas pelos utilizadores.

No NoHR Web para além das definições já provenientes do *WebProtégé* relativamente às contas do utilizadores ou aos seus projetos, foi necessário adicionar definições referentes ao NoHR, existentes também na versão *desktop*, como as definições relativamente ao raciocinador de execução das perguntas e definições sobre os mapeamentos para as bases de dados.

As definições pertencentes ao NoHR, tanto do raciocinador como dos mapeamentos para bases de dados estão associadas a cada projeto e não à conta do utilizador. Desta forma o utilizador pode ter vários projetos com definições personalizadas para cada um deles.

```
{
    "_id":{"$oid":"5e19f7b0f8871f3fdccf68ff"},
    "ProjectID":"44380ba7-bdbf-4d26-b8d2-83a2db1c2377",
    "DLInferenceEngineEL":true,
    "DLInferenceEngineQL":true,
    "DLInferenceEngineRL":true,
    "DLInferenceEngine":"HERMIT"
}
```

Listagem 3.2: Exemplo de um documento que contém as definições do raciocinador para um projeto específico

3.3.2.1 Definições do raciocinador do NoHR

Nas definições do Raciocinador, o utilizador pode indicar o raciocinador que pretende usar. Embora o perfil seja selecionado pela ontologia, o utilizador tem a opção de selecionar um raciocinador, *HermiT* ou o *Konclude*, para utilizar em vez do perfil predeterminado.

Na figura 3.2 está o exemplo de um documento que guarda as definições do raciocinador do NoHR para um determinado projeto identificado pelo seu identificador único, onde o perfil *EL*, *QL* e o *RL* são guardados em valores booleanos que indicam se será usado um raciocinador genérico, no caso de ter o valor *"true"*, ou se será usado o módulo dedicado para o perfil predeterminado pela ontologia no caso de haver algum, como mencionado no capítulo 2, caso o valor seja *"false"*. O raciocinador, *HermiT* ou *Konclude*, é guardado numa *String* que o representa.

3.3.2.2 Definições dos mapeamentos de bases de dados do NoHR

De forma a que um utilizador possa usar a funcionalidade de mapeamentos para bases de dados, é obrigatório fornecer o acesso à base de dados através do *driver ODBC*.

Na figura 3.3 está representada a estrutura de um documento que guarda informação sobre os mapeamentos para as bases de dados para um determinado projeto identificado pelo seu identificador único. Como um utilizador pode usar vários *drivers ODBC* no processo de execução de perguntas, a estrutura contempla uma lista bidimensional de *Strings*. Desta forma, são guardadas várias ligações *ODBC*. Cada driver *ODBC* contém a ligação a uma base de dados onde o utilizador terá de fornecer qual o tipo e o nome da base de dados, assim como as suas credenciais de acesso.

3.3.3 Mapeamentos de bases de dados no MongoDB

Para guardar os mapeamentos de bases de dados é usada a *collection* **NoHRDatabase-Mappings** que tem uma estrutura semelhante à *collection* que guarda as regras não-monotónicas.

```
{
    "_id":{"$oid":"5e2a1eb760084416a891649a"},
    "ProjectID":"678d8083-7593-483f-843b-8844a27cabec",
    "dbSettings":
    [
        [
            "odbc.ini",
            "database_name",
            "MySQL",
            "rootuser",
            "rootpass"
        ],
        [
            "odbc2.ini",
            "database_name2",
            "MySQL",
            "rootuser",
            "rootpass"
        ]
    ]
}
```

Listagem 3.3: Exemplo de um documento que contém as definições dos mapeamentos para bases de dados de um projeto específico

Os documentos da *collection* **NoHRDatabaseMappings** são constituídos por um identificador único do documento, o identificador único do projeto e uma lista de mapeamento para as bases de dados no formato de uma lista de *Strings*.

Ao contrário das regras não-monotónicas que podem facilmente ser convertidas para uma *String* que contem toda a informação da regra, os mapeamentos para bases de dados têm uma estrutura mais complexa. Por isso é necessário converter todo o objeto para uma sintaxe possível de guardar num ficheiro, em que a informação possa ficar toda numa *String*. Esta funcionalidade já existe na versão *desktop* do NoHR, sendo usada para guardar os mapeamentos de bases de dados em ficheiros. No NoHR Web esta funcionalidade foi também usada para guardar os mapeamentos no *MongoDB*.

Na listagem 3.4 é possível observar a estrutura de um documento que guarda os mapeamentos para bases de dados. Neste exemplo, apenas está contemplada uma *String* na lista, ou seja, apenas um mapeamento, que está indentado para facilitar a sua visualização. Um mapeamento contém uma pesquisa numa base de dados relacional em que o utilizador pode escolher as colunas e as tabelas que pretende usar, incluindo fazer *Joins* entre tabelas selecionando as colunas onde pretende fazer *Join*. Ainda sobre a listagem 3.4, no exemplo que está apresentado é selecionada uma coluna, denominada *column1*, da tabela *Table1*, com o *driver* ODBC *odbc.ini* e o predicado *test*.

Para definir um mapeamento o utilizador tem primeiro de definir o *driver ODBC*, como explicado na secção 3.3.2.2, de forma a fornecer as suas credenciais para efetuar a ligação a uma base de dados.

```
{
    "_id":{"$oid":"5e66391d75d2257a7d196f2a"},
    "ProjectID":"678d8083-7593-483f-843b-8844a27cabec",
    "dbMappings":
    [
       "<mapping>odbc.ini<mapping>
        <mapping>
           <table>
              <tableInfo>Table1<tableInfo>
              <tableInfo>t1<tableInfo>
              <tableInfo>
                  <tableJoinOnPairs>
                      <col><col>
                  <tableJoinOnPairs>
                      <col><col>
                  <tableJoinOnPairs>
              <tableInfo>
           <table>
        <mapping>
        <column>
            <colInfo>
              Table1 as t1
              <colInfo>t1
              <colInfo>Column1
              <colInfo>false
            <colInfo>
        <column>
        <mapping>
            <predicate>
              test
              <predicate>1
            <predicate>
        <mapping>"
    ]
}
```

Listagem 3.4: Exemplo de um documento de mapeamentos para bases de dados do MongoDB, este exemplo apenas contém um mapeamento

Toda essa informação fica armazenada numa *String* que contém uma estrutura específica para armazenar mapeamentos.

3.4 Integração do NoHR Web com o sistema de permissões

No *WebProtégé* é permitido que os utilizadores façam partilha de projetos entre si, isto significa que o mesmo projeto pode ser visível para vários utilizadores. Mas nem sempre o dono do projeto quer que os outros utilizadores tenham os mesmos privilégios que ele.

Para permitir que um utilizador possa escolher as permissões que outros utilizadores tenham no seu projeto, o *WebProtégé* tem um sistema de permissões integrado que funciona por *roles*.

Existem quatro *roles* no *WebProtégé* cada uma com níveis diferentes de permissões, que estão ordenadas pelo nível de permissão que incluí, por exemplo, a *role* 3 da lista

contém as permissões das *roles* 1 e 2.

1. **VIEW** Neste nível de permissão , os utilizadores apenas podem visualizar o projeto sem terem autorização para efetuar qualquer alteração no mesmo.

2. **COMMENT** Esta *role* permite que os utilizadores para além de terem as permissões da *role* 1 possam também deixar comentários no projeto.

3. **EDIT** Esta *role* para além de incluir as duas anteriores, permite aos utilizadores fazerem alterações no projeto.

4. **MANAGE** Esta *role* dá a um utilizador as permissões totais sobre o projeto.

Como já referido, visto que o *WebProtégé* contempla esta funcionalidade de permissões, no NoHR Web, as funcionalidades do NoHR foram integradas neste sistema de permissões na *role* que melhor se adequava à sua função.

Dito isto, no NoHR Web, um utilizador que apenas tenha a permissão de visualização do projeto, apenas pode executar operações que não façam qualquer alteração no projeto, como visualizar as várias páginas, descarregar informação ou executar perguntas, que também não altera o estado do projeto. Caso tenha a permissão de edição, o utilizador poderá editar a informação referente ao NoHR, como demonstrado na tabela 3.1.

No *WebProtégé*, apenas os utilizadores que têm a permissão de gestão do projeto (*Manage*), podem alterar as definições de um projeto. Como as definições do NoHR estão inseridas na estrutura de definições de projetos no *WebProtégé*, seguem o mesmo comportamento das restantes definições.

	VIEW	**COMMENT**	**EDIT**	**MANAGE**
View Rules	X	X	X	X
Create Rules			X	X
Delete Rules			X	X
Upload Rules			X	X
Download Rules	X	X	X	X
View DBMappings	X	X	X	X
Create DBMappings			X	X
Delete DBMappings			X	X
Upload DBMappings			X	X
Download DBMappings	X	X	X	X
Execute Queries	X	X	X	X
View DBMappings Settings				X
Edit DBMappings Settings				X
View NoHR reasoner Settings				X
Edit NoHR reasoner Settings				X

Tabela 3.1: Permissões inerentes a cada *role* para a execução das funcionalidades do NoHR

A tabela 3.1 mostra, para cada *role*, que funcionalidades do NoHR podem ser executadas pelos utilizadores nos projetos partilhados.

3.5 Integração com múltiplos utilizadores

No NoHR *desktop* apenas existe uma instância do raciocinador da aplicação, que é inicializada quando o utilizador abre a aba de perguntas do NoHR no *Protégé*. A instância do raciocinador é então inicializada com a ontologia do projeto carregada pelo utilizador através de um ficheiro, o respetivo vocabulário da ontologia, um conjunto de regras não-monotónicas carregado também pelo utilizador através de um ficheiro que contém um conjunto de regras, um conjunto de mapeamentos para bases de dados, carregado também através de um ficheiro e as preferências na execução de perguntas, tais como se o utilizador tem preferência em usar um raciocinador, como mencionado no capítulo 2, como também definições para efetuar ligações com bases de dados, onde os utilizadores podem definir os *drivers* ODBC.

Como o *plug-in* NoHR é utilizado na aplicação *Protégé*, que corre na máquina do utilizador em *localhost* em que apenas existe um utilizador, não há necessidade de manter ativa mais do que uma instância do raciocinador do NoHR.

Para a aplicação do NoHR Web este paradigma mudou, ou seja, já não existe apenas um utilizador que vai usar as funcionalidades do NoHR e para isso foi necessário desenvolver uma solução para permitir a integração dos vários utilizadores às funcionalidades do NoHR em simultâneo. Para implementar esta solução foi criada toda uma estrutura na aplicação NoHR Web em que foram criadas estruturas de dados com o objetivo de gerir a integração de várias instâncias de raciocinadores do NoHR.

A figura 3.2 contém a estrutura de um objeto de uma instância do raciocinador do NoHR. Estas instâncias são guardadas em estruturas de dados dinâmicas associadas aos utilizadores e aos seus projetos como será explicado na secção 3.5.1. As instâncias do raciocinador do NoHR estão inseridas na estrutura da aplicação web como está representado na figura 3.3, em que cada utilizador terá uma instância como a da figura 3.2 para cada um dos seus projetos sempre que executar uma pergunta. Cada instância do objeto do raciocinador do NoHR tem uma instância do NoHR, que engloba a estrutura usada no NoHR *desktop* e é inicializada passando a ontologia do projeto, o vocabulário, o conjunto de regras não-monotónicas e os mapeamentos para bases de dados, tendo também instâncias próprias do Interprolog e do XSB. A instância do objeto do raciocinador do NoHR guarda também o *parser* da pergunta para a ontologia do projeto e uma estrutura que faz o mapeamento das respostas para o utilizador como será explicado na secção 3.7.1.

A figura 3.3 representa a estrutura do NoHR Web, detalhando a integração com o NoHR. De forma a simplificar a figura, cada bloco *"NoHR Instance"* contêm a figura 3.2. Existe uma instância do raciocinador do NoHR por cada projeto que cada utilizador possa aceder.

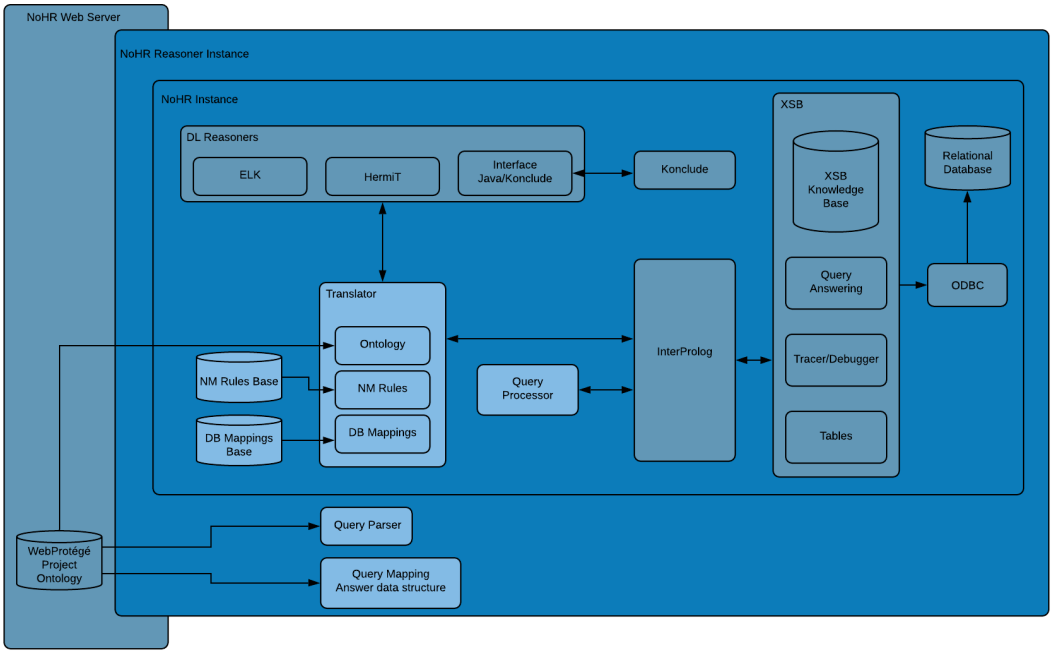

Figura 3.2: Estrutura de uma instância do raciocinador no NoHR Web

As instâncias do raciocinador do NoHR são criadas quando um utilizador executa uma pergunta no projeto, caso não exista já uma instância ativa, carregando a ontologia associada ao projeto e as regras não-monotónicas armazenadas no *MongoDB*. Pela figura 3.3 é possível perceber que os utilizadores podem partilhar projetos mantendo instâncias do raciocinador do NoHR independentes como será explicado de seguida.

3.5.1 Estrutura de gestão de Instâncias do raciocinador do NoHR

Foi criada na aplicação uma estrutura de dados com a finalidade de fazer a gestão das várias instâncias de raciocinadores que possam estar ativas em simultâneo.

Map<username, Map<project, NoHR Reasoner Instance> >

O mapa exterior tem como chave uma string que representa o nome do utilizador, visto que não pode haver dois utilizadores com o mesmo nome na aplicação, e tem como valor um outro mapa que tem como chave o identificador do projeto e como valor uma instância do objeto do raciocinador do NoHR, igual ao da figura 3.2, desta forma cada utilizador pode ter uma instância para cada um dos seus projetos. A decisão estrutural de atribuir as instâncias dos raciocinadores por utilizador e por projeto e não apenas por projeto, foi para permitir que vários utilizadores possam correr perguntas no mesmo projeto sem interferências por parte de outros utilizadores, visto que os utilizadores podem partilhar

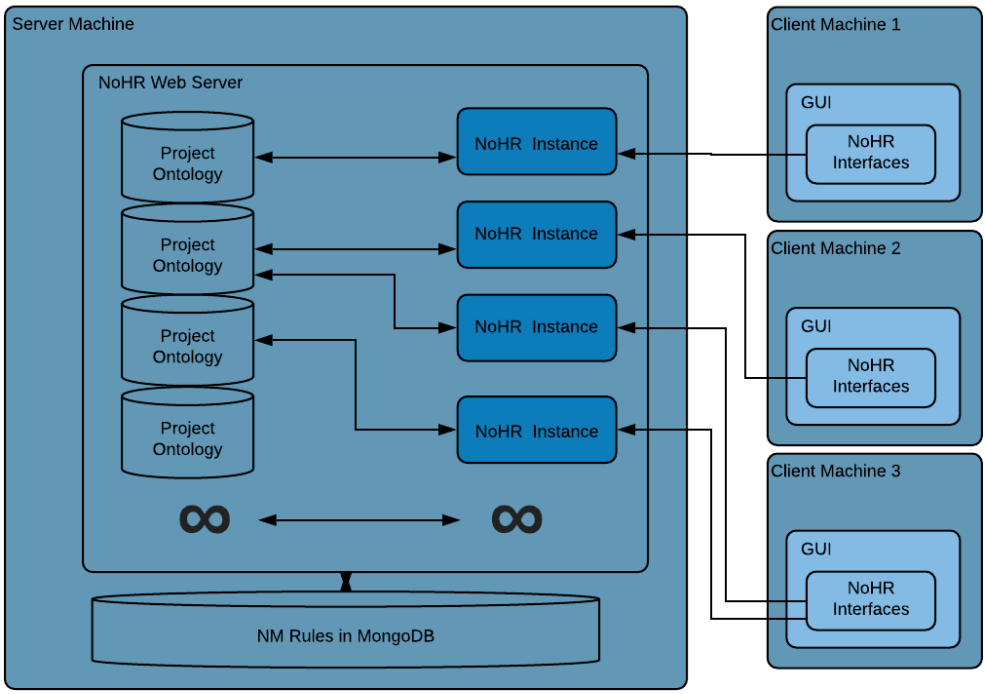

Figura 3.3: Estrutura da aplicação NoHR Web

projetos. Desta forma os utilizadores têm a sua própria instância do raciocinador, mesmo que estejam a trabalhar simultaneamente no mesmo projeto com outro utilizador.

Esta instância do objeto *NoHR Reasoner Instance*, contém o raciocinador do NoHR , igual ao do NoHR *desktop*, o *parser* da pergunta especifica para a ontologia e uma estrutura que faz o mapeamento das respostas para o utilizador. Esta última será explicada mais à frente na secção 3.7.1.

Sempre que um utilizador executa uma pergunta, caso não tenha uma instância do raciocinador inicializada, será inicializada uma e guardada na estrutura acima, com o respetivo nome de utilizador e identificador de projeto como chaves. Para cada novo projeto do mesmo utilizador em que uma instância do raciocinador seja inicializada, será adicionada à estrutura com o identificador do projeto ao nome já existente do utilizador como chaves do mapa interior e do mapa exterior respetivamente.

Na versão *desktop* do NoHR, uma instância apenas é terminada caso sejam efetuadas alterações na ontologia, nas regras ou nas configurações do raciocinador. De outro modo, a instância permanece aberta entre a execução de várias perguntas.

Numa aplicação web, é necessário ter atenção ao comportamento dos utilizadores e aos recursos do servidor, e por isso no NoHR Web, para além das já referidas formas de terminar uma instância do raciocinador, é necessário ter em atenção mais aspetos. No caso do utilizador terminar a sua sessão na aplicação, a sua sessão terminar devido

a inatividade, ou o utilizador não efetuar qualquer pergunta durante um período de tempo predeterminado na configuração do servidor. Uma instância do raciocinador será terminada devido a todos estes motivos apresentados, poupando desta forma os recursos do servidor. A instância do raciocinador é posteriormente retirada da estrutura de dados, de forma a evitar que esta estrutura cresça infinitamente.

3.5.2 Estrutura de temporizadores de Instância do raciocinador do NoHR

Como foi referido na secção 3.5.1, uma instância do raciocinador do NoHR pode ser terminada se o utilizador não efetuar qualquer pergunta enquanto uma dada instância está ativa. Para fazer essa verificação foi criada uma estrutura que contabiliza o tempo que cada instância tem desde a última pergunta efetuada ou desde que foi inicializada.

Map<username, Map<project, Timer> >

Esta estrutura de dados guarda os temporizadores de cada instância ativa do NoHR, ou seja, introduz um temporizador quando uma instância é inicializada.

O tempo para terminar as instâncias do raciocinador pode ser configurado através do ficheiro de propriedades do *WebProtégé* em que o gestor do servidor deve introduzir o tempo em minutos que pretende dar a cada instância do raciocinador.

Se uma instância estiver inativa durante o período de tempo predefinido no ficheiro, o temporizador irá terminar a instância do raciocinador e removê-la da estrutura apresentada na secção 3.5.1. A cada pergunta executada numa instância do raciocinador, o temporizador para essa instância será reinicializado, para evitar que uma instância do raciocinador seja terminada enquanto está em utilização.

Esta estrutura de dados guarda informação por utilizadores e por projetos. Um temporizador associado a um utilizador e a um projeto, é um temporizador para o mesmo utilizador e projeto na estrutura de dados que guarda as instâncias do raciocinador.

Desta forma, é feita uma gestão mais eficiente das instâncias ativas do raciocinador, juntando aos restantes mecanismos de gestão de utilização das instâncias do raciocinador no servidor. E permite que o gestor do servidor possa gerir o tempo que as instâncias do raciocinador permanecem ativas de uma forma acessível.

3.5.3 Estrutura de gestão de execução de perguntas

A estrutura de dados anterior, assegura que cada utilizador tenha uma instância do objeto do raciocinador do NoHR para cada um dos seus projetos. Ainda assim, é preciso garantir que um utilizador não execute duas perguntas em simultâneo no mesmo projeto, ou seja, duas perguntas em simultâneo na mesma instância do raciocinador do NoHR, ou irá causar erros no servidor.

Para controlar esta situação foi criada uma estrutura de dados para manter atualizado que instâncias do NoHR estão a executar perguntas.

Map<username, Map<project, Boolean> >

Esta estrutura é semelhante à anterior, com a exceção de em vez de guardar as instâncias do objeto do raciocinador do NoHR, guarda valores booleanos que indicam se uma pergunta está a executar, verdadeiro, ou não, falso, para um dado projeto de um utilizador. Estas duas estruturas de dados apresentadas estão sempre sincronizadas uma com a outra, ou seja, as estruturas terão as mesmas entradas a referir os mesmos utilizadores e projetos de forma coerente.

Quando um utilizador inicia a execução de uma pergunta, é consultada a estrutura de dados acima de forma a verificar se o utilizador tem alguma pergunta em execução no respetivo projeto, não executando a pergunta caso exista outra em execução. Caso contrário, a execução prossegue e é guardado o valor booleano *verdade* na estrutura de dados acima durante toda a execução da pergunta, de forma a impedir que o mesmo utilizador possa correr outra pergunta durante a execução da pergunta anterior. Quando a pergunta termina é guardado o valor booleano *falso* permitindo então a execução de novas perguntas.

Para não permitir leituras sujas na estrutura de dados acima entre a verificação e a escrita na estrutura de dados, é necessário tornar as operações de verificação e escrita atómicas, de forma a prevenir que duas execuções de perguntas do mesmo utilizador no mesmo projeto façam a verificação de execução exatamente ao mesmo tempo e conseguissem executar a pergunta, isto causaria um erro no servidor.

Para prevenir esta situação foram utilizados *locks*. Antes de um utilizador efetuar uma leitura nesta estrutura é feito um *lock*, e só é feito *unlock* após a respetiva escrita na estrutura de dados ou caso não tenha permissão para efetuar a escrita. Desta forma um utilizador só pode fazer uma leitura se o estado se encontrar *unlock*, que significa que já foi feito uma escrita na estrutura de dados. Como já referido, esta situação só pode acontecer no caso do mesmo utilizador tentar correr duas perguntas no mesmo projeto em simultâneo.

Para prevenir que esta estrutura cresça de forma indefinida como a estrutura de dados que guarda instâncias do raciocinadores, sempre que uma instância é terminada na outra estrutura de dados, é também removida desta estrutura, de forma a manter ambas as estruturas consistentes e apenas com a informação necessária.

Por motivos de eficiência foi criada uma outra estrutura para permitir que verificações em que seja necessário procurar por um projeto possam ser executadas de forma mais eficiente. As estruturas de dados anteriores não tornam a tarefa de pesquisa por projetos eficiente, por ser necessário passar por todos os utilizadores para procurar por um projeto.

Map<project, Boolean>

Este mapa contém como chave o identificador do projeto e como valor um booleano. Se o valor booleano associado a um projeto for *"true"*, significa que existe uma pergunta do

NoHR em execução no projeto, caso contrário, não existe nenhuma pergunta em execução no projeto.

Com esta estrutura de dados, é mais eficiente verificar se um projeto tem uma pergunta em execução. Esta verificação é útil quando os recursos de um projeto são libertados do servidor por inatividade e é necessário verificar se existe alguma pergunta em execução nesse projeto. No caso de existir uma pergunta em execução, a instância do raciocinador do NoHR não é terminada, caso contrário, será terminada.

3.6 Comunicação entre o cliente e o servidor

As comunicações entre o servidor e o cliente do *WebProtégé* usam o serviço *Remote Procedure Call* do GWT, que permite transferir objetos Java entre o cliente e o servidor sobre o protocolo HTTP[3]. Como o GWT integra a arquitetura AJAX, em que a diferença destas aplicações para as tradicionais aplicações web em HTML é que as aplicações AJAX não necessitam de recarregar a página HTML durante a execução. As páginas das aplicações AJAX funcionam como aplicações dentro do *browser* e desta forma não é necessário fazer um pedido ao servidor para fazer atualizações na interface do cliente.

A implementação do serviço RPC[4] do *Google Web Toolkit* é baseada na conhecida arquitetura Java *Servlet*, que é uma componente de software Java que implementa as funcionalidades de um servidor. No código do cliente, é usada uma classe *proxy* gerada automaticamente pelo GWT que efetua os pedidos ao serviço, encarregando-se da serialização e deserialização dos objetos, os argumentos nos métodos e o valor retornado.

É necessário ter em conta que serviços RPC não são iguais aos serviços web baseados em SOAP[5] ou REST[6], que são protocolos para a troca de informação na implementação de serviços web. O RPC é apenas uma forma simples de transferir informação entre o cliente e o servidor, inerente ao Google Web Toolkit.

No *WebProtégé*, os pedidos do cliente são encapsulados em classes denominadas de *"action"*, em que para cada tipo de operação que um cliente pretenda executar no servidor existe uma classe ação própria. Para as respostas do servidor de volta para o cliente a informação é encapsulada em classes denominadas de *"result"*.

No caso do *WebProtégé*, quando um pedido de uma ação específica chega ao servidor, é necessário tratar essa informação de forma diferente para cada ação. No servidor para cada ação, existe um *"handler"*, que trata da informação enviada pelo cliente através da classe ação específica e envia a resposta através da classe resultado específica também da mesma ação, podemos considerar estas classes como um triplo, *action*, *handler* e *result*, para cada operação diferente.

[3] Hypertext Transfer Protocol
[4] Remote Procedure Call
[5] Simple Object Access Protocol
[6] Representational State Transfer

Na implementação das novas funcionalidades no NoHR Web, foram criadas classes *"action"*, *"result"* e *"handler"* para cada funcionalidade do NoHR em que seja necessário trocar informação com o servidor. Por exemplo, quando um cliente executa uma pergunta, a *String* que contém a pergunta é encapsulada numa instância da classe ação, onde a pergunta será tratada na classe *handler* e as respostas serão retornadas para o cliente através da classe *result*.

Para a implementação do NoHR Web foi utilizada a técnica comunicação usada no *WebProtégé*. Para todas as novas operações do NoHR Web foi necessário criar as classes *"action"*, *"handler"* e *"result"* de forma a tratar cada operação do utilizador de forma diferente no servidor.

3.7 Comunicação do servidor sobre o estado operações

Num caso normal, o cliente será capaz de comunicar com o servidor e receber as suas respostas, ainda assim, no caso do NoHR Web, é necessário saber se a operação do utilizador teve sucesso no servidor ou se ocorreu algum erro no servidor durante o processamento da informação. Por exemplo, um utilizador pode introduzir uma regra não-monotónica com um erro de sintaxe, em que nesse caso o servidor terá de informar o cliente que a regra não foi inserida. Poderá ocorrer também algum erro na instância do raciocinador do NoHR durante o processamento de uma pergunta ou o utilizador submeter um ficheiro de regras não-monotónicas em que uma ou várias regras contêm erros de sintaxe. Todos estes erros ocorrem no servidor e o mesmo terá de informar o cliente que não foi possível realizar a operação.

De forma a que o servidor possa informar o cliente do estado da operação, foi adicionado um enumerado que contém vários códigos de estados que podem ter ocorrido na operação. Estes códigos, vão desde mensagens de erro de sintaxe de regras não-monotónicas e perguntas, até a erros que possam ter ocorrido na instância do raciocinador do NoHR ou erros de comunicação com a base de dados *MongoDB*.

Este código é enviado na resposta do cliente ao servidor, no caso da operação ter sido efetuada com sucesso será enviado um *OK*, caso contrário será enviado um código que representa o possível erro que tenha surgido no servidor. O cliente com este código fará o tratamento da exceção como será apresentado no capítulo 4.

3.7.1 Execução de Perguntas

Quando um utilizador executa uma pergunta, é enviado um pedido ao servidor que contém a pergunta introduzida pelo utilizador na interface.

O servidor começa por verificar se o utilizador tem outra pergunta em execução no mesmo projeto e caso tenha uma pergunta em execução, é retornada uma mensagem ao utilizador e a pergunta não será executada, caso contrário a execução prossegue naturalmente. Esta verificação é feita utilizando *locks* de forma a prevenir leituras sujas na

estrutura de dados, como referido na secção 3.5.3.

Caso o utilizador não tenha uma instância do raciocinador para o projeto onde a pergunta está a ser executada, será criado um *parser* para a pergunta que contém a ontologia especifica em que o utilizador está a trabalhar, e uma instância do raciocinador através dos mecanismos explicados anteriormente com as regras não monótonicas definas previamente pelo utilizador, a ontologia do projeto, o respetivo vocabulário, as definições do raciocinador do NoHR e os mapeamentos para bases de dados.

Com a instância criada, é então executada a pergunta pelo raciocinador do NoHR e as respostas são então retornadas por uma instância do programa externo XSB para o servidor. A conexão entre o XSB e o servidor do *WebProtégé* que contém o NoHR é feita igualmente através do Interprolog como na versão *desktop* do NoHR.

De seguida as respostas do XSB terão de ser mapeadas antes de serem devolvidas ao utilizador. Contrariamente ao que acontece nas ontologias desenvolvidas usando o *Protégé*, em que os nomes das classes, indivíduos e propriedades estão contidos nos IRIs[7] dos elementos da ontologia, no *WebProtégé* os nomes dos elementos estão contidos numa *rdf:label*. Desta forma, depois de obter as repostas às perguntas é necessário fazer o mapeamento entre os IRIs com as *RDF labels* de todos os elementos da ontologia, de forma a associar cada IRI a uma *RDF label* de cada elemento da ontologia.

Map<IRI , rdf:label>

Quando uma instância é inicializada, o servidor fará uma pesquisa pelas *rdf:labels* de todas as classes, indivíduos e atributos e irá associar essa informação à IRI do elemento específico que é a chave da estrutura de dados referida em cima. Numa ontologia desenvolvida usando o *Protégé*, o nome que o utilizador fornece a cada elemento fica presente na IRI desse elemento. Já numa ontologia criada no *WebProtégé*, o nome que identifica o elemento da ontologia não está presente na IRI do elemento. Desta forma, como o XSB devolve as IRI dos elementos, é necessário recorrer à estrutura de dados para obter o valor da *rdf:label* correspondente a cada IRI que esteja contida nas repostas do XSB, para que o utilizador reconheça nas respostas os elementos presentes na ontologia.

Para as ontologias desenvolvidas usando o *Protégé*, em que o identificador do elemento da ontologia, que consiste num nome mais curto dado pelo utilizador para identificar o elemento, fica inserido no final do IRI do elemento, separado por um caractere especial definido pelo utilizador ('#','/' ou ':'). Para este caso, o raciocinador do NoHR já tem presente o mecanismo para separar o identificador do elemento do respetivo IRI quando devolve as respostas.

Desta forma, para manter a compatibilidade com ontologias criadas usando o *Protégé*, quando o raciocinador retorna o identificador do elemento diretamente, é enviado nesse mesmo formato nas respostas para o cliente.

[7] Internationalized Resource Identifier

Enquanto a instância do raciocinador do NoHR permanecer ativa, o raciocinador, o *parser* da pergunta e os mapeamentos dos *rdf:labels* ficam guardados na estrutura de dados, que guarda instâncias do raciocinador do NoHR, associados a um projeto e a um utilizador.

Capítulo 4

Estrutura do cliente da aplicação

Neste capítulo vai ser introduzida a interface desenvolvida para a aplicação NoHR Web, onde iremos explicar todas as novas janelas da interfaces e serão apresentadas todas a funcionalidades e a forma como deve ser feita essa interação.

4.1 Introdução da nova interface

Como foi referido anteriormente, o NoHR Web divide-se num servidor e num cliente. O lado do cliente é composto por todo o código que corre em *JavaScript* no *web browser* do utilizador.

O servidor é o responsável por enviar a página HTML que contém a informação pretendida pelos clientes, embora com o detalhe de que quando o cliente efetua alterações, o servidor apenas envia informação sobre as alterações para que o cliente não tenha de recarregar a página toda a cada pedido ao servidor, mas sim, o que realmente foi alterado. Isto reflete-se numa interface mais dinâmica e consequentemente não carrega a rede com informação que o cliente já possui.

A interface da aplicação web é assíncrona, significa que se o utilizador fizer um pedido ao servidor, a interface não ficará bloqueada à espera da resposta do servidor. Quando a resposta chegar do servidor, serão feitas as atualizações na interface.

No *WebProtégé* a interface referente aos projetos é constituída por uma estrutura de abas, em que dentro de cada aba é permitido ao utilizador configurar as janelas que integram a aba. Por norma, cada aba tem as janelas já definidas em que o utilizador pode sempre alterá-las e repô-las se preferir as janelas originais.

De forma a integrar as funcionalidades do NoHR *desktop* no novo NoHR Web, foram criadas abas que contêm janelas que integram a interface da aplicação. No NoHR Web foram adicionadas três novas abas à interface da aplicação, nomeadamente, as abas

NoHR Rules, *NoHR Query* e a *NoHR Database Mappings* que iremos abordar mais à frente, mantendo a mesma nomenclatura do NoHR *desktop*.

Figura 4.1: Abas padrão do NoHR Web

Dentro destas abas adicionadas ao NoHR Web, foram criadas janelas para permitir que os utilizadores interajam com as funcionalidades do NoHR. Uma aba contém um conjunto de janelas definidas e ajustadas pelo utilizador, apesar de quando o utilizador inicia um projeto já vem com abas predefinidas que contêm janelas.

As janelas do NoHR criadas para o NoHR Web foram, a janela que contém uma lista de regras não-monotónicas e permite executar operações sobre as regras, a janela de perguntas que permite ao utilizador fazer perguntas sobre a ontologia e um conjunto de regras não-monotónicas, em que as respostas serão apresentadas numa lista, e a janela de mapeamentos para bases de dados que permite aos utilizadores definirem os mapeamentos. Essas janelas serão apresentadas mais em detalhe nas secções seguintes. Estas janelas estão inseridas nas três abas mencionadas anteriormente.

Para além destas três janelas, foram adicionadas secções na página de definições de projetos do NoHR Web que permitem configurar o raciocinador do NoHR e a configurar os *drivers* ODBC para a ligação a bases de dados.

4.1.1 Gestão das janelas do NoHR

No *WebProtégé*, e consequentemente no NoHR Web, cada projeto tem as abas previamente definidas, sendo que o utilizador tem liberdade de as apagar e criar novas abas onde pode escolher o nome que pretender. O utilizador pode configurar qualquer aba da aplicação, mesmo as abas predefinidas, adicionado qualquer janela da aplicação, podendo ser instâncias da mesma janela. Esta funcionalidade torna a interface da aplicação extremamente flexível e permite que cada utilizador a possa configurar a seu gosto.

O facto do utilizador poder adicionar duas instâncias da mesma janela à mesma aba pode ser útil caso o utilizador pretenda ver os resultados de duas perguntas distintas na mesma aba, em para isso terá de colocar duas janelas de perguntas na mesma aba e executar uma pergunta em cada uma delas. No caso da janela que mostra as regras não-monotónicas e a janela que mostra os mapeamentos para bases de dados foi criada uma estrutura para manter todas as instâncias destas janelas com a mesma informação em todos os momentos, isto porque, se o utilizador adicionar uma regra não-monotónica numa instância de uma janela de regras não-monotónicas, têm de ser atualizadas todas as instâncias das mesma janela com a mesma informação.

Para manter a consistência entre as janelas foi criada uma estrutura que guarda todas as instâncias de janelas do NoHR Web. Essa estrutura contém uma lista que guarda todas as instâncias de uma janela ativas, eliminando-as quando o utilizador as fecha de forma a manter a lista coerente com as janelas existentes.

Desta forma se o utilizador fizer alterações no projeto através de uma das janelas, todas as instâncias referentes à mesma janela serão atualizadas com a mesma informação de modo a manter a informação das janelas consistente, exceto para as instâncias da janela *NoHR Query*, para permitir a visualização de resultados de várias perguntas distintas se o utilizador assim o quiser.

Da mesma forma, enquanto uma pergunta está em execução numa janela de perguntas, todas as instâncias da janela de perguntas têm o botão de execução desativado porque não é permitido executar uma pergunta enquanto outra está a correr.

As interfaces de regras não-monotónicas, de perguntas e de mapeamentos para as bases de dados contêm esta estrutura de gestão para as suas janelas, pois estas interfaces estão integradas na estrutura de janelas já existente do *WebProtégé* como será explicado de seguida.

4.1.2 Interface de regras

Tal como no NoHR *desktop*, existe uma interface no NoHR Web onde os utilizadores podem definir as regras não-monotónicas de um projeto.

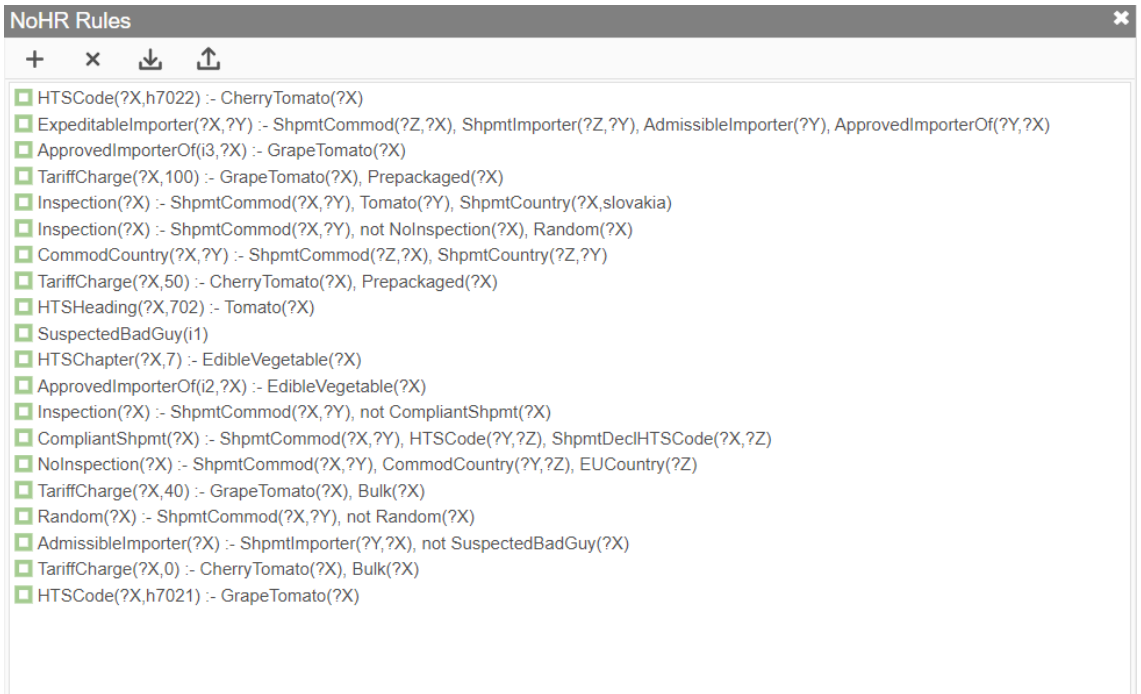

Figura 4.2: Interface de regras não-monotónicas do NoHR Web

Esta interface apresenta uma lista que contém todas as regras não-monotónicas adicionadas pelo utilizador ao respetivo projeto, em que o utilizador pode interagir usando os botões localizados no canto superior esquerdo da janela de regras.

Os quatro botões representados na parte superior da figura 4.2, permitem ao utilizador criar, apagar, baixar um ficheiro para o seu dispositivo que contém o conjunto de regras não-monotónicas e carregar do seu dispositivo um ficheiro com um conjunto de regras

não-monotónicas para o NoHR Web. Existe ainda uma quinta operação que não tem nenhuma botão associado, que é a operação que permite editar uma regra não monotónica já existente no projeto.

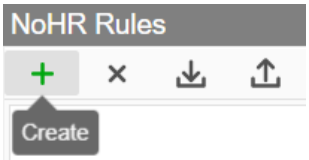

Figura 4.3: Botão que abre formulário de criação de uma regras não-monotónicas

Ao carregar no botão *"Create"*, irá surgir uma janela que permite ao utilizador introduzir uma nova regra não monotónica.

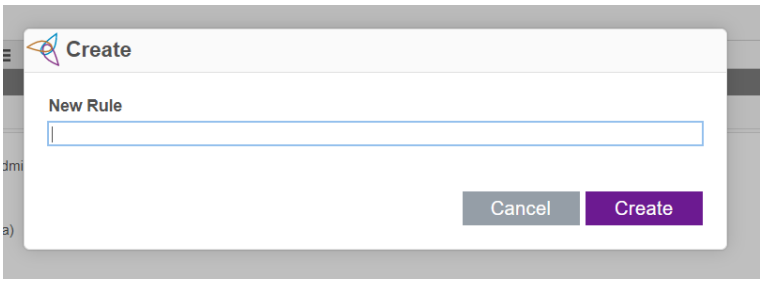

Figura 4.4: Formulário para criação de uma regra não monotónica

No formulário da figura 4.4, o utilizador pode escrever a regra não monotónica que pretende introduzir na aplicação, a regra será enviada para o servidor onde será avaliada pelo *parser* do raciocinador e só será aceite se respeitar a sintaxe do NoHR, caso contrário, a aplicação mostrará ao utilizador uma mensagem que informará que a regra contém um erro e a mesma não será inserida.

No caso do utilizador pretender editar uma regra já existente no projeto, o utilizador deve tocar rapidamente duas vezes sobre uma regra existente na lista de regras não-monotónicas. Ao fazer isso, irá surgir uma janela que permite atualizar uma regra não monotónica já existente.

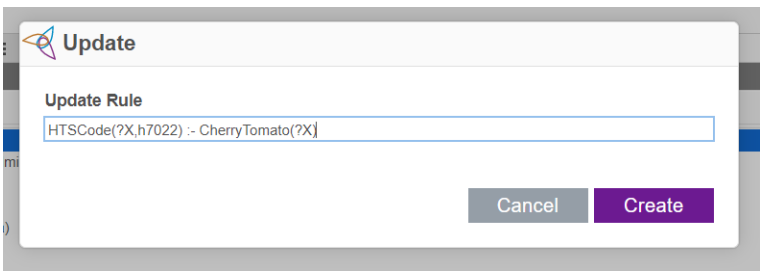

Figura 4.5: Formulário para atualização de uma regra não monotónica

Na janela da figura 4.5, a caixa de texto que permite ao utilizador editar uma regra não monotónica já vem preenchida com uma *String* que representa a regra que o utilizador pretende alterar, de forma a facilitar a atualização da mesma pelo utilizador. Quando

submetida, a nova regra do utilizador será também avaliada pelo *parser* do raciocinador e só será feita a alteração caso não tenha nenhum erro de sintaxe.

Para apagar uma ou várias regras não-monotónicas, o utilizador deve selecionar as regras que pretende apagar na lista de regras e de seguida pressionar o botão *"Delete"* na janela. Para eliminar várias regras de uma vez, o utilizador deve selecionar um conjunto de regras não-monotónicas e pressionar o botão *"Delete"*.

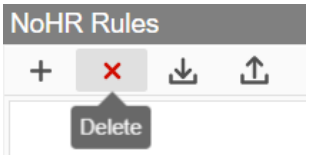

Figura 4.6: Botão que permite apagar uma ou várias regras não-monotónicas

As regras serão eliminadas de todas as vistas de regras não-monotónicas do NoHR Web assim como do servidor.

O NoHR Web permite também que um utilizador possa descarregar as regras não-monotónicas de um respetivo projeto. Quando o utilizador tocar no botão *"Download"* na janela, será descarregado um ficheiro com a extensão *".nohr"* através do *browser*. Este ficheiro contém o conjunto de regras não-monotónicas, uma por linha, do respetivo projeto.

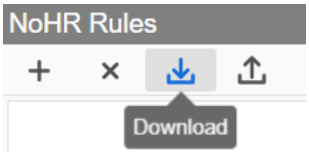

Figura 4.7: Botão que permite descarregar o conjunto de regras não-monotónicas do respetivo projeto

Para permitir que um utilizador possa inserir um conjunto de regras não-monotónicas num determinado projeto (em vez de estar a criar regra a regra que seria um processo monótono e demorado), existe a possibilidade de carregar ficheiros de regras não-monotónicas com a extensão *".nohr"* para um projeto. Esse ficheiro deve ter uma regra não monotónica por linha, igual à estrutura do ficheiro que se obtém quando se descarrega um conjunto de regras.

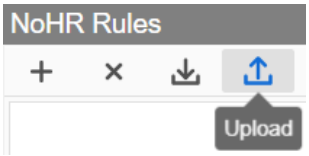

Figura 4.8: Botão que permite abrir um formulário para carregar um ficheiro de regras não-monotónicas para um determinado projeto

A tocar no botão de *"Upload"*, a aplicação mostrará uma janela onde é possível ao

utilizador selecionar o ficheiro de regras não-monotónicas que pretende carregar para o projeto.

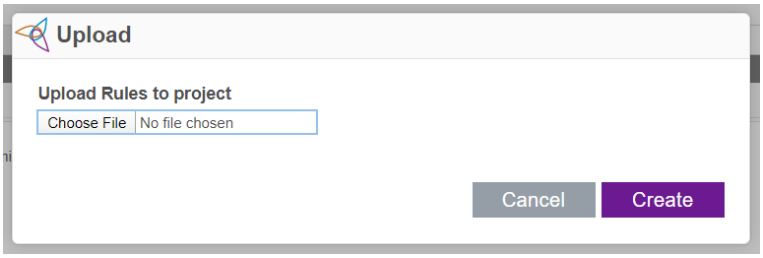

Figura 4.9: Formulário que permite carregar um ficheiro de regras não-monotónicas para introduzir no projeto

Na janela da figura 4.9, o utilizador deve selecionar um ficheiro selecionando a opção *Choose File* na interface, isto fará abrir uma janela que contém o gestor de ficheiros do dispositivo do utilizador onde este poderá selecionar o ficheiro pretendido. Quando executada a operação de *"Upload"*, todas as regras não-monotónicas existentes no projeto serão substituídas pelas regras não-monotónicas contidas no ficheiro inserido.

4.1.3 Interface de perguntas

O NoHR Web, tal como o NoHR *desktop*, contém uma interface que permite aos utilizadores executarem perguntas sobre a ontologia e sobre um conjunto de regras não-monotónicas.

Na página inicial do projeto, existe uma aba concebida para o NoHR Web, denominada *NoHR Query*, que contém três janelas. Uma janela que contém a hierarquia de classes da ontologia do projeto, uma janela que contém o conjunto das regras não-monotónicas do projeto e uma janela para a execução de perguntas.

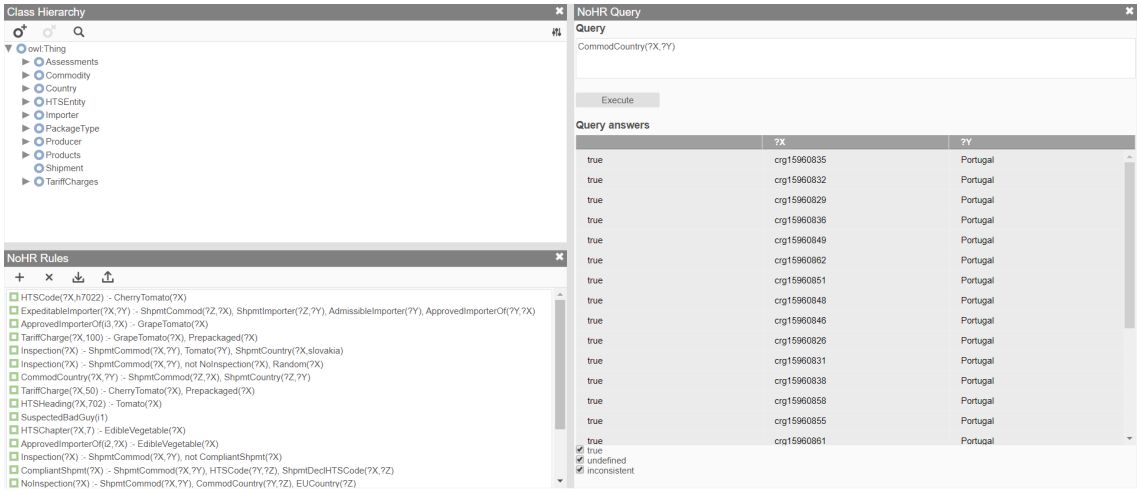

Figura 4.10: Aba *NoHR Query* do NoHR Web

A janela que contém a hierarquia de classes é uma interface nativa do *WebProtégé* e a janela que contém as regras não-monotónicas do NoHR é igual à apresentada na secção 4.1.2.

A janela do lado direito na figura 4.10 é a interface que permite os utilizadores executarem perguntas sobre um conjunto de regras não-monotónicas e uma ontologia.

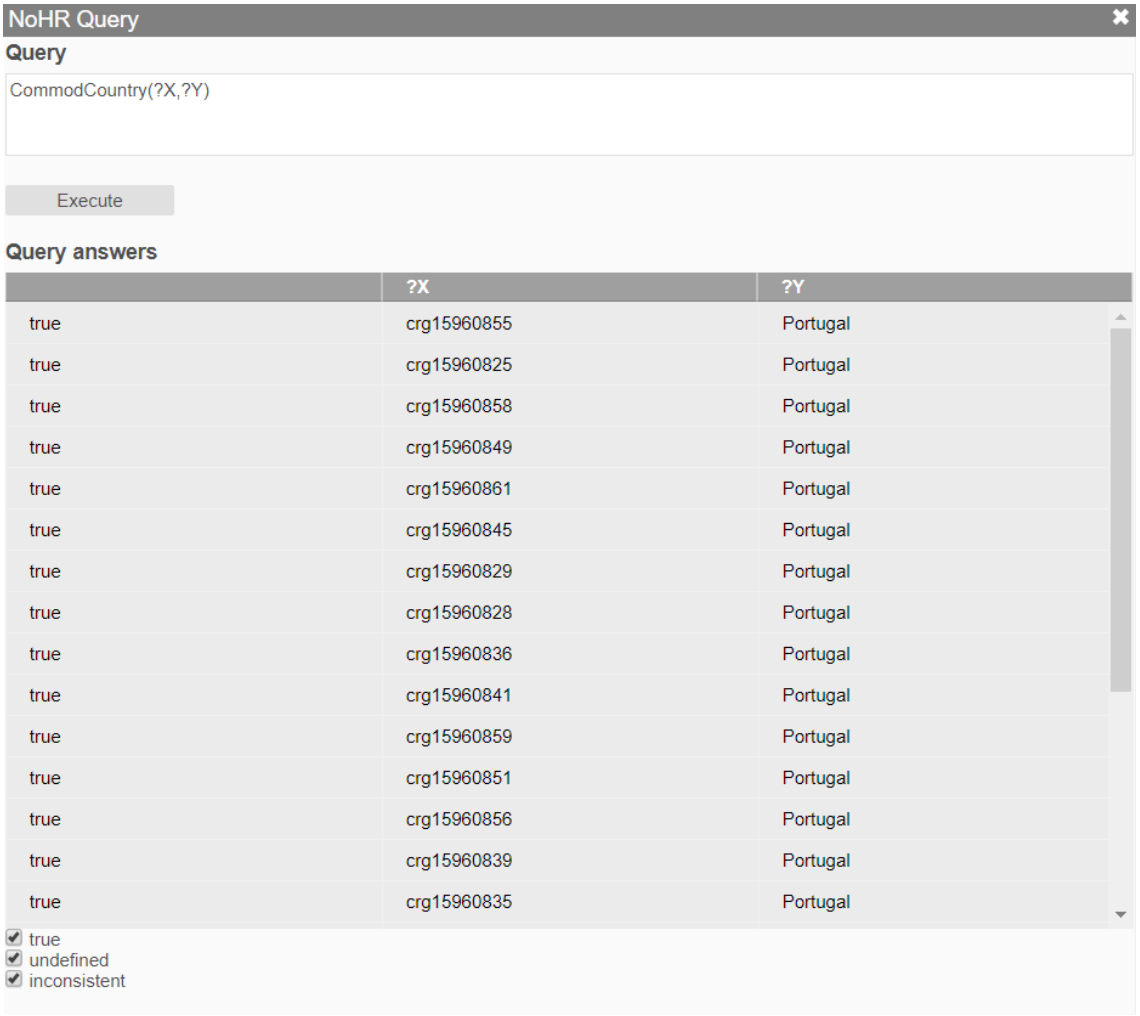

Figura 4.11: Janela *NoHR Query* do NoHR Web

Quando um utilizador pressiona o botão *"Execute"* na interface, é enviado um pedido ao servidor que contém a pergunta inserida pelo utilizador. Caso a pergunta não respeite a sintaxe das regras não-monotónicas do NoHR, avaliada através do parser do raciocinador, será enviada uma mensagem ao utilizador com a informação do erro e a pergunta não será executada nem a instância do raciocinador será inicializada.

O servidor irá de seguida verificar se o utilizador possui uma instância do raciocinador do NoHR para este projeto e criará uma caso não exista nenhuma associada ao projeto. Essa instância será usada para processar a pergunta do utilizador e devolver as respostas assim que possível.

Enquanto o servidor está a processar a pergunta do utilizador, será apresentado na

interface um símbolo indicando que o servidor está a processar o pedido. Desta forma o utilizador tem informação visual que o seu pedido está a ser processado pelo servidor, caso contrário o utilizador não saberia se tinha submetido a sua pergunta com sucesso.

As respostas são apresentadas numa tabela em que a primeira coluna contém os valores das avaliações para cada substituição de variáveis, que podem ter o valor *"verdadeiro"*, *"indefinido"* ou *"inconsistente"*. O número de restantes colunas depende da quantidade de variáveis introduzidas pelo utilizador na pergunta inserida.

O utilizador pode também filtrar o número de respostas que pretende receber por tipo de avaliação através das *checkboxes* em baixo da tabela, ou seja, se o utilizador apenas mantiver ativa a *checkbox* correspondente ao valor *"verdade"*, apenas serão apresentadas as respostas cuja substituição de variáveis tenham a avaliação *"verdadeira"*. O utilizador tem de ter pelo menos uma das *checkboxes* selecionadas, embora a própria interface nunca permita que o utilizador desmarque todas as *checkboxes*.

4.1.4 Interface de mapeamentos para a ligação a bases de dados

No caso dos mapeamentos para bases de dados, foi adicionado ao NoHR Web uma aba que contém uma janela que permite ao utilizador o acesso rápido às funcionalidades dos mapeamentos para bases de dados.

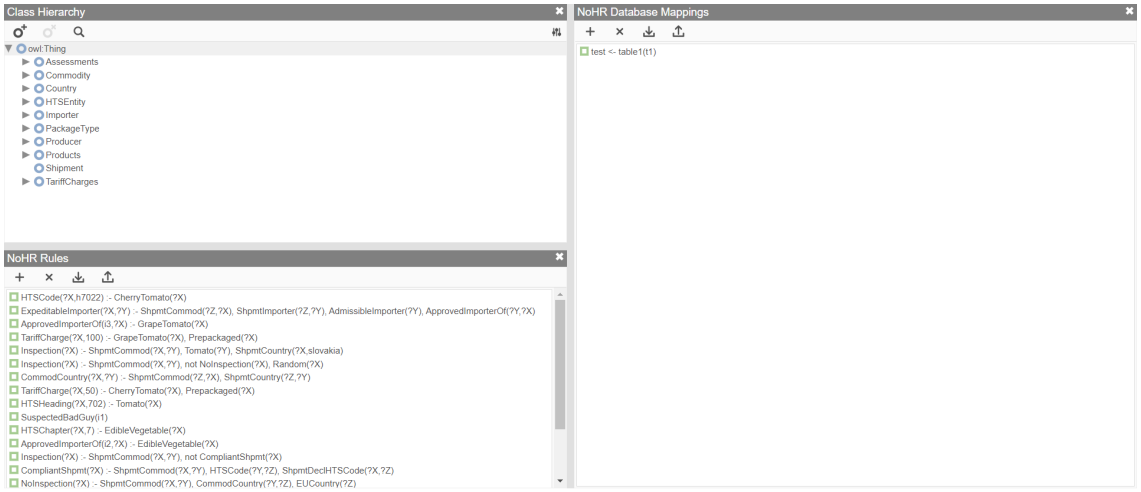

Figura 4.12: Aba *NoHR Database Mappings* do NoHR Web

Esta aba tem uma estrutura semelhante à aba *"NoHR Query"* apresentada na secção 4.1.3, com a diferença que a janela de perguntas foi substituída pela janela de mapeamentos para as bases de dados no lado direito da figura 4.12 assim coma na versão *desktop* do NoHR.

A janela de mapeamentos para bases de dados permite que o utilizador adicione, atualize, elimine, descarregue e carregue mapeamentos de bases de dados para o NoHR Web, assim como na interface de regras não-monotónicas onde foram apresentados os botões que permitem estas funcionalidades.

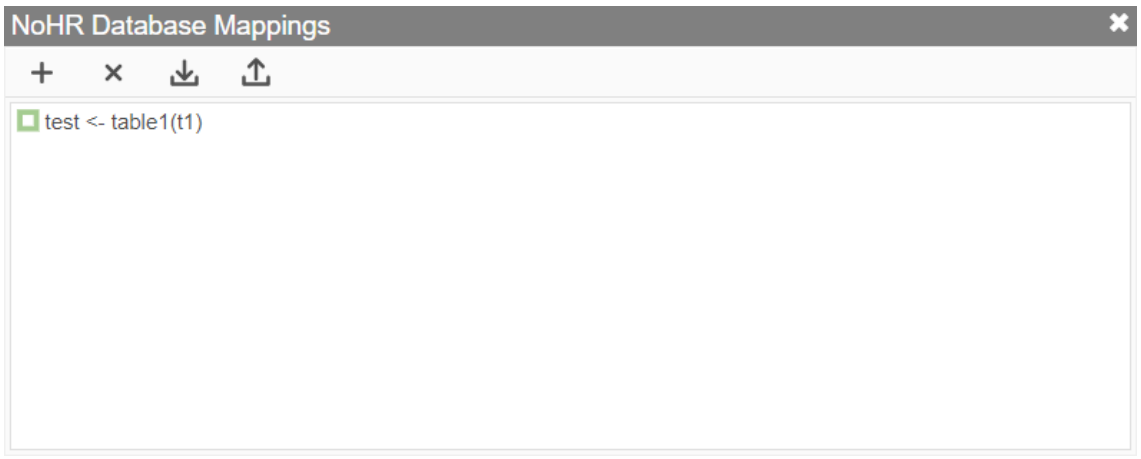

Figura 4.13: Aba *NoHR Database Mappings* do NoHR Web

Quando o utilizador pressiona o botão para adicionar um novo mapeamento de bases de dados, representado pelo símbolo de "mais", surgirá um formulário para o utilizador preencher com a informação necessária.

No formulário da figura 4.14 o utilizador deve selecionar o *driver* ODBC, que será apresentado na secção 4.1.5 de que forma o utilizador poderá configurar estes *drivers*. Posteriormente, o utilizador pode escolher as tabelas e as colunas que pretende selecionar dessas tabelas, e até mesmo fazer operações de *"Join"* entre as tabelas através da interface, como será explicado de seguida. Está também disponível o modo de inserção manual de uma pesquisa SQL, onde para isso o utilizador deve pressionar o botão *"Write SQL"*.

A primeira tabela da figura 4.14, o campo *Select Columns*, é referente às colunas que o utilizador pretende selecionar. Sempre que o utilizador pressionar o botão *add* referente a esta tabela, surgirá um formulário para a criação da coluna na tabela de colunas. Embora, não seja permitido adicionar uma coluna sem que antes o utilizador tenha adicionado um elemento na segunda tabela, e por isso a interface não o permite.

Na figura 4.15, o utilizador poderá introduzir uma coluna de uma tabela previamente criada.

A segunda tabela da figura 4.14, o campo *From Tables*, é referente às tabelas das bases de dados. Sempre que o utilizador pressionar o botão *"add"* para adicionar um elemento a esta tabela, surgirá um formulário de criação de tabelas.

Neste formulário o utilizador pode introduzir a tabela e selecionar se pretende que a tabela faça a operação de *"Join"* com uma outra tabela previamente introduzida pelo utilizador e que colunas de cada tabela serão usadas para efetuar a operação de *"Join"*. O utilizador deve também introduzir o campo *"INTO Predicate"* para definir o predicado do mapeamento de bases de dados no NoHR Web.

Os campos *"Arity"* e *"SQL"* são preenchidos automaticamente à medida que o utilizador vai editando o formulário da figura 4.14. Caso o utilizador pretenda editar estes campos manualmente, deve pressionar o botão *"Write SQL"*, no fundo do formulário o que fará desbloquear estes campos. Assim o utilizador será livre de escrever o seu próprio

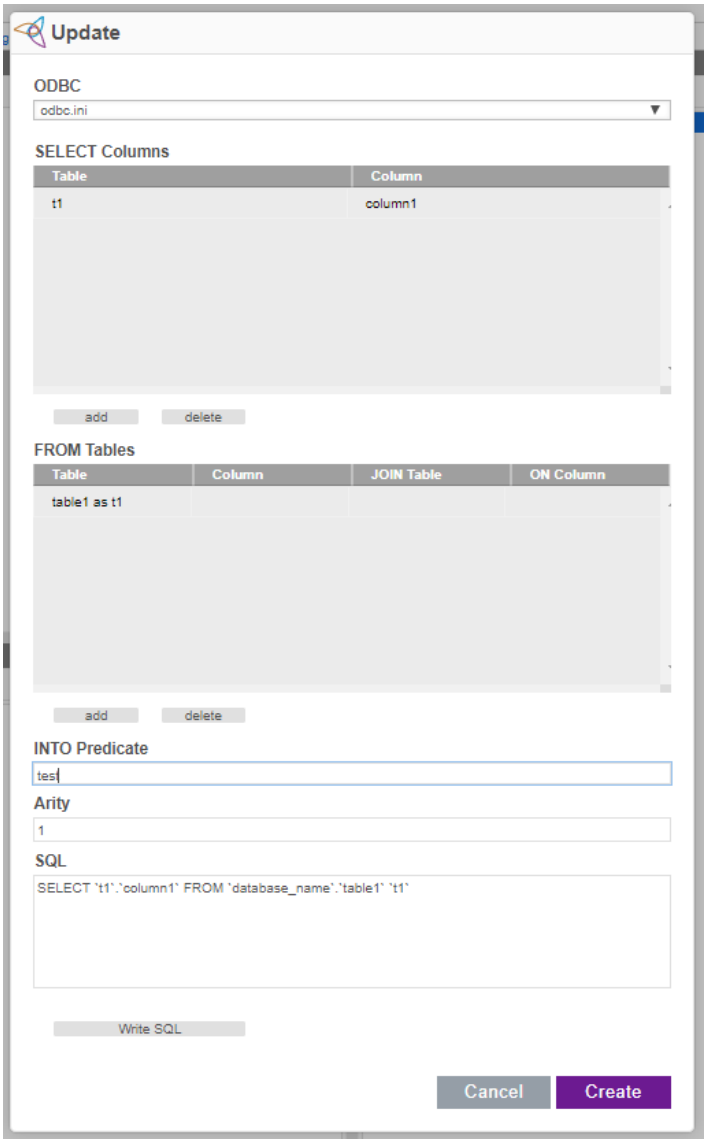

Figura 4.14: Formulário para criação de mapeamentos de bases de dados no NoHR Web

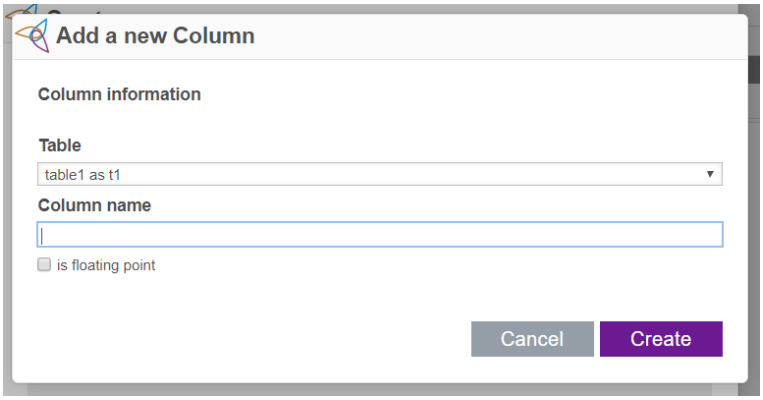

Figura 4.15: Formulário para criação de colunas na tabela *"Select Columns"* no NoHR Web

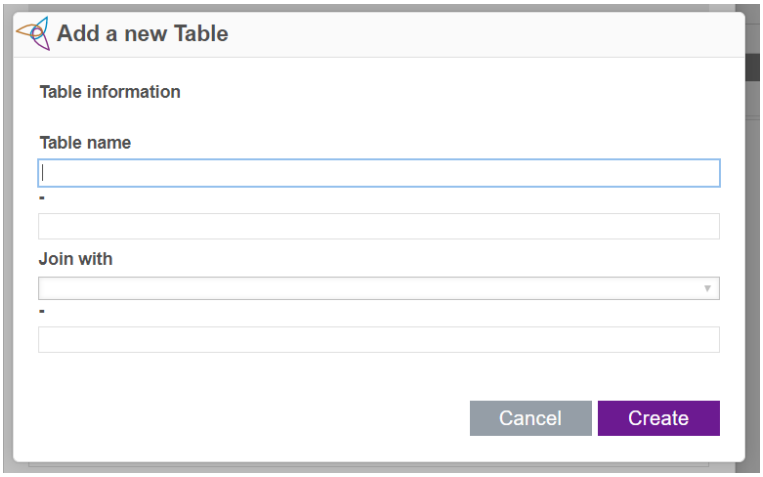

Figura 4.16: Formulário para criação de tabelas na tabela *"From Tables"* no NoHR Web

código *SQL* e escolher a aridade. Neste modo as tabelas estão bloqueadas e o utilizador não poderá introduzir nenhuma elemento nas tabelas *"Select Columns"* e *"From Tables"*.

4.1.5 Interfaces de definições

O *WebProtégé* contém uma interface para o utilizador configurar os seus projetos. No NoHR Web foram adicionadas vistas à janela de definições do projeto para que o utilizador possa configurar os aspetos relacionados com o NoHR.

Nas definições do projeto, foi adicionada uma vista de definições do NoHR que permite ao utilizador escolher se pretende usar um raciocinador genérico na execução das suas perguntas sobre o perfil predeterminado para a ontologia contida no projeto, exatamente como na versão *desktop* do NoHR.

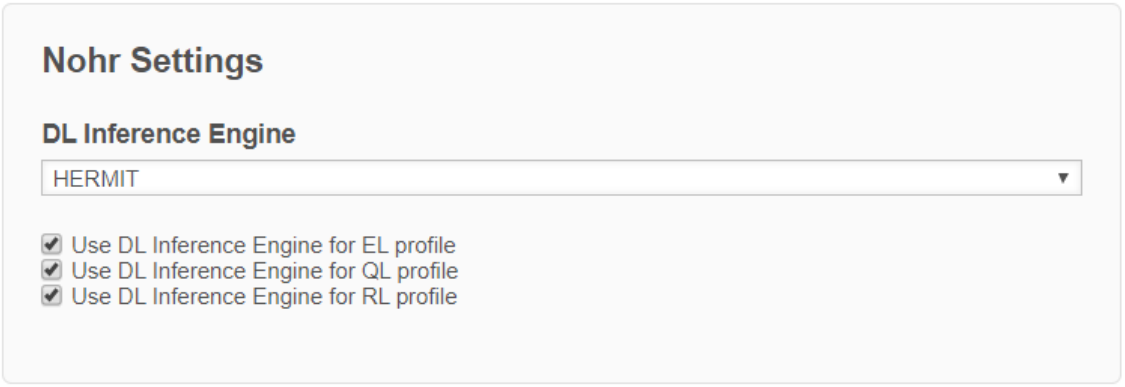

Figura 4.17: Vista de definições do raciocinador do NoHR no NoHR Web

Nesta vista o utilizador pode escolher para quais perfis quer usar o motor de inferência geral, *Hermit* ou *Konclude*, em vez do módulo próprio.

Para além da vista de definições do raciocinador do NoHR, existe também uma vista para configurar a ligação a bases de dados. Esta vista é um pouco diferente da anterior,

visto que permite que o utilizador possa criar várias instâncias da mesma vista de forma a poder configurar várias ligações a diferentes bases de dados. A vista da figura 4.18, contém um botão no fundo da vista que permite ao utilizador criar outra secção da mesma vista. Este botão está sempre disponível na última secção da interface para que o utilizador possa continuar a adicionar secções.

Para o utilizador poder criar mapeamentos de bases dados, têm primeiro de fornecer o acesso ao DBMS[1], à qual pretende aceder, através do *driver* ODBC[2].

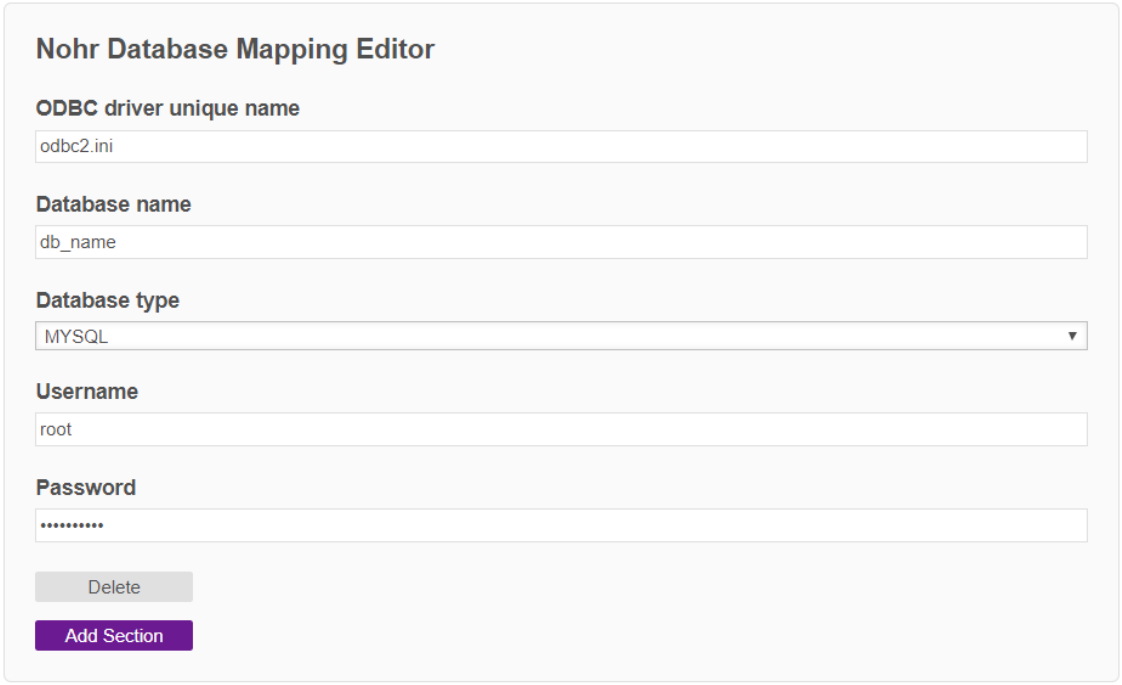

Figura 4.18: Vista de definições do raciocinador do NoHR no NoHR Web

Esta vista é constituída por cinco campos, onde o utilizador deve configurar a sua ligação à base de dados fornecendo o driver ODBC, o nome e o tipo da base de dados e as sua credenciais para a ligação, estando o campo para a palavra chave com proteção de texto. Como o utilizador pode usar diferentes *drivers* ODBC para o mesmo processo de raciocínio, é possível adicionar outro *driver* ODBC, em que para isso o utilizador deve tocar no botão *Add Section*, situado no fundo da figura 4.18 para surgir outra vista igual a da figura 4.18 onde o utilizador poderá configurar um novo *driver* ODBC.

4.1.6 Janelas informativas da operação

Para além das janelas já apresentadas, existem ainda janelas de *popup* que informam os utilizadores caso exista alguma operação que não tenha corrido bem ou não tenha sido efetuada no servidor. Como por exemplo uma regra não-monotónica ou uma pergunta que não tenha a sintaxe correta.

[1] Database Management System
[2] Open Database Connectivity

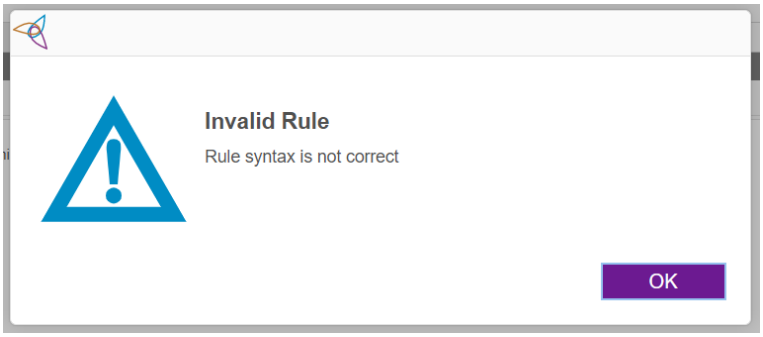

Figura 4.19: Janela de *popup* que informa o utilizador que a regra não monotónica inserida tem um erro de sintaxe

Como mencionado no capítulo 3, o cliente irá analisar o código que o servidor enviou para cada operação e agir em conformidade com o estado do servidor. No caso de ter ocorrido algum erro, o cliente irá exibir uma janela que informa o erro específico que ocorreu no servidor, embora nem todas as mensagens de erro tenham de vir do servidor. Por exemplo, se o utilizador submeter um ficheiro de regras não-monotónicas com a extensão do ficheiro incorreta ou tentar submeter um ficheiro sem ter selecionado nenhum, neste caso irá surgir uma janela de erro sem ser feito nenhum pedido ao servidor.

Para além de informarem ao utilizador sobre possíveis erros, estas janelas impedem algumas operações indesejadas pelos utilizadores, como por exemplo, um utilizador eliminar uma regra não monotónica sem o querer fazer. Para evitar estas situações, aparecerá uma janela onde o utilizador poderá confirmar ou cancelar a operação. Desta forma, existe mais segurança para evitar possíveis operações indesejadas pelos utilizadores.

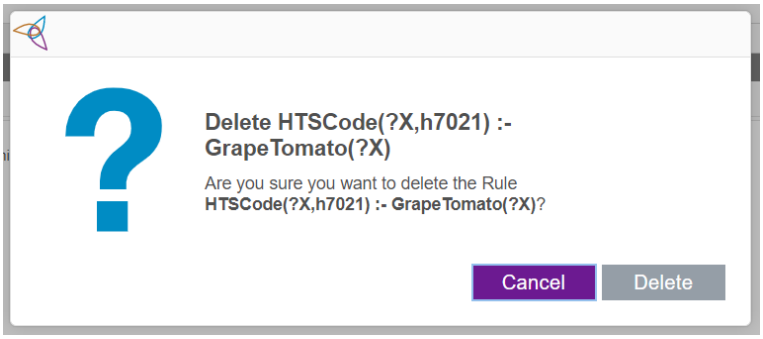

Figura 4.20: Janela de *popup* que permite ao utilizador confirmar se pretende eliminar uma regra não monotónica

4.1.7 Observações

Resumindo, a interface do NoHR, desenvolvida para o NoHR Web teve por base a interface já existente do NoHR *desktop* mas não ficou por aí. Foi implementada de forma a melhorar a interação com o utilizador, mantendo a apresentação do *WebProtégé* de forma a estar visualmente integrada com o mesmo, para que todos os utilizadores do NoHR que já

conheçam o *WebProtégé*, ou não, possam rapidamente adaptar-se à nova interface.

As janelas criadas para o NoHR estão também elas inseridas na estrutura de abas do NoHR Web e essas janelas podem ser inseridas em qualquer uma outra aba da aplicação, o utilizador não precisa de ficar preso às abas originais da aplicação.

Colocando os botões de cada janela na numa barra superior, o utilizador pode de forma rápida passar o cursor por cima de cada um deles para saber a funcionalidade de cada botão. Visualmente, foi mantida a estrutura usada nas restantes janelas do *WebProtégé*, para que o utilizador consiga interagir no NoHR Web da mesma forma em todas as janelas.

Capítulo 5

Avaliação e Resultados

Neste capítulo foi comparado o desempenho do NoHR desktop com o NoHR Web em tarefas tais como, a inicialização de instâncias do raciocinador para diferentes ontologias, o desempenho no processo de raciocínio e no caso do NoHR Web a utilização em simultâneo por vários utilizadores na realização das tarefas anteriores

5.1 Avaliação de desempenho do NoHR Desktop e do NoHR Web

Os motivos pelos quais se optou por integrar o NoHR no *WebProtégé*, para além da facilidade de utilização disponibilizada aos utilizadores através de uma aplicação web, foi para que a máquina do utilizador não comprometesse o poder de raciocínio do NoHR. Considerando que um servidor terá um poder computacional maior que o dispositivo do utilizador, em teoria a eficiência de raciocínio num servidor será maior. Por outro lado, temos de ter em conta a taxa de utilização de um servidor, em que se estiver a ser sobrecarregado de pedidos poderá não conseguir responder de forma eficiente a todos eles, e neste caso, o desempenho seria inferior comparativamente com o raciocínio numa máquina local. Embora a solução apresentada retire do utilizador todo o trabalho de configurar o editor na sua máquina, como acontece com o *Protégé*. Isto permite que os utilizadores possam começar logo a trabalhar num projeto sem que seja necessário mais nenhum procedimento.

Após a apresentação de todas as novas características do NoHR Web, é necessário fazer uma comparação detalhada com a versão existente do NoHR, de forma a testar a eficiência da nova solução comparativamente com a anterior.

Inicialmente, vão ser comparados os dois editores de ontologias, o *Protégé* e o *WebProtégé* na eficiência em carregar as ontologias do utilizador para os respetivos editores

através de ficheiros, em que todos os testes serão realizados utilizando o *Protégé* na versão 5.2.0 e o *WebProtégé* na versão 4.0.0.

De forma a obter valores reais da integração do raciocinador do NoHR na web, vão ser comparados os tempos de inicialização do raciocinador do NoHR *desktop* com o NoHR Web, utilizando apenas um utilizador neste último, desta forma teremos uma comparação real de ambas as aplicações. Apesar das funcionalidades do NoHR web terem sido implementadas tendo em conta questões de eficiência e considerando a gestão de recursos para a utilização do servidor pelos vários utilizadores, o servidor tem também o seu limite de recursos. Por esta razão, vão ser feitos testes de sobrecarga ao servidor utilizando 5, 10 e 25 utilizadores em simultâneo a executar perguntas ao servidor, onde serão retirados tempos da inicialização dos raciocinadores e os tempos das perguntas.

Todos os testes que se seguem foram realizados na mesma máquina, com o sistema operativo Windows 10 e com um processador Intel i7 de 3.40 GHz e 12gb de memória Ram dedicada para o servidor do NoHR Web e para o NoHR *desktop*, de forma a que fosse possível tirar conclusões sobre a eficiência usando o mesmo *hardware*. No caso do NoHR Web, o servidor foi testado com um cliente na mesma máquina, exceto nos testes de sobrecarga, onde os pedidos do cliente foram executados a partir de outra máquina com o sistema operativo Windows 10 e com um processador Intel i7 de 2.60 GHz e 12gb de memória Ram.

Nos testes que se seguem, foi feita a média sobre o número de utilizadores em teste, utilizando o tempo de cada utilizador, tirando desta forma o tempo de execução dos processos em análise.

5.2 Tempo de carregamento de Ontologias no WebProtégé vs Protégé

Para avaliar as novas condições do NoHR na web, é necessário comparar o *WebProtégé* com o *Protégé*, neste caso é necessário fazer um comparação do tempo que ambos os editores demoram a carregar uma ontologia para a aplicação através de um ficheiro.

Os testes foram realizados utilizando o *Protégé* na versão 5.2.0 e o *WebProtégé* na versão 4.0.0 beta 3.

Na tabela 5.1, são apresentados os tempos de carregamento de ontologias no *Protégé* e no *WebProtégé*. As primeiras 3 colunas da tabela representam o nome da ontologia, o tamanho do ficheiro em *Megabytes* e o número de axiomas da ontologia, respetivamente. A coluna "*Protégé*", contém os tempos de carregamento de ontologias no *Protégé*. A coluna "Criar WP", representa o tempo de carregamento de uma ontologia através da criação de um projeto, enquanto a coluna "Abrir WP"representa o tempo de instanciação de uma ontologia num projeto já existente.

Nesta comparação do tempo de carregamento de ontologias entre o *Protégé* e o *WebProtégé* é possível observar que em alguns casos o *WebProtégé* é mais rápido a efetuar o

Ontologias	Tamanho	Nº de axiomas	Protégé	Criar WP	Abrir WP
SnomedCT	109.2 mb	1 355 008	21.408s	18.913s	8.089s
Union	106.8 mb	859 512	11.831s	8.003s	2.645s
Snomed Anatomy	53.801 mb	661 680	8.120s	7.962s	1.458s
Fma	34.6 mb	496 172	7.978s	4.500s	1.412s
Galen8	128.8 mb	168 466	13.024s	8.648s	3.368s
go2	11.3 mb	109 853	3.050s	1.176s	0.304s
Chebi	5.0 mb	98 383	3.101s	0.994s	0.309s
Molecule Role	13.0 mb	93 595	2.604s	0.689s	0.263s
go1	11.6 mb	89 370	2.663s	0.927s	0.361s
Galen	10.2 mb	60 633	2.863s	0.857s	0.262s
Fly Anatomy	3.302 mb	27 064	2.048s	1.831s	0.178s

Tabela 5.1: Comparação do tempo de carregamento e de instanciação de ontologias no *Protégé* e no *WebProtégé* em segundos

carregamento de ontologias através de ficheiros até um fator de 4, embora no caso do *WebProtégé*, o carregamento de uma ontologia através de um ficheiro só seja necessário fazer na criação do projeto, a partir daí a ontologia só tem de ser instanciada sempre que o utilizador abrir o projeto, o que tem um tempo ainda inferior ao de carregamento através de um ficheiro.

5.3 Eficiência dos raciocinadores do NoHR Web

Umas das diferenças do NoHR Web para o NoHR *desktop* são as bibliotecas utilizadas pelos raciocinadores. No NoHR *desktop* é usada a biblioteca owlapi-osgidistribution do repositório OWLAPI enquando no *WebProtégé* é utilizada a biblioteca owlapi-apibinding que engloba as bibliotecas owlapi-api, owlapi-impl, owlapi-parsers, owlapi-oboformat, owlapi-tools, owlapi-fixers e owlapi-rio do repositório OWLAPI.

Existe contudo, um problema de eficiência encontrado na versão da biblioteca owlapi-apibinding usada no *WebProtégé* . Na versão *desktop* do NoHR é utilizada a versão 4.2.6 do OWLAPI, enquanto as últimas versões do *WebProtégé* utilizam as versão 4.5.13. Esta perda de eficiência é sentida na inferência de uma ontologia no momento da inicialização dos raciocinadores HermiT e ELK, utilizando a versão 4.5.13 do owlapi-apibinding.

Foram testadas várias versões da biblioteca owlapi-apibinding, e reparou-se que a eficiência da inferência de ontologias ia piorando nas versões mais recentes.

Este aumento dos tempos de processamento de ontologias, deve-se ao aumento da complexidade de um método da biblioteca OWLAPI responsável pela inferência da ontologia por parte do ELK e do HermiT, em que é possível reparar que nas versões mais recentes, devido ao aumento da complexidade são responsáveis por este aumento dos tempos. Como é possível observar na tabela 5.2, entre a versão 4.2.6, usada na versão *desktop* do NoHR, e a versão 4.5.13, utilizada no *WebProtégé* e consequentemente no NoHR

Ontologias	OWLAPI 4.2.6	OWLAPI 4.3.1	OWLAPI 4.5.13
Fly_Anatomy	1.6 sec	2.3 sec	5.0 sec
Chebi	3.1 sec	25.6 sec	47.6 sec
Snomed CT	117 sec	-	-

Tabela 5.2: Comparação de tempo de pré processamento, ou seja, classificação e tradução, de ontologias utilizando o raciocinador ELK nas diferentes versões do OWLAPI em segundos

Web, o tempo de processamento aumenta por um fator de 15, em que no caso do Snomed CT, após 2 horas, o processo de inferência foi manualmente interrompido para as versões 4.3.1 e 4.5.13, onde é possível observar uma diferença de 2 minutos para cerca de 2 horas por um aumento por um fator de 60.

Dentro do repositório do OWLAPI existem várias bibliotecas, estando este método responsável pelo aumento dos tempos de inferência localizado na biblioteca owlapi-api. Para resolver esta situação, foi alterado o ficheiro jar da versão 4.5.13 da biblioteca owlapi-api para incluir o método usado pelos raciocinadores do NoHR da versão 4.2.6. Esse ficheiro modificado foi posteriormente usado na compilação do NoHR Web e desta forma foram restaurados os tempos normais de inferência de ontologias nos raciocinadores.

5.4 Eficiência do raciocínio do NoHR Web comparativamente com o Nohr desktop

Posteriormente, foram realizados testes apenas à eficiência do raciocinador, ou seja, utilizando apenas um utilizador no NoHR Web de forma a fazer uma análise limpa do desempenho do raciocinador quando comparado com o NoHR *desktop*. Para isso foram utilizadas várias ontologias, em que foi comparado o tempo de inicialização do raciocinador, dividido no tempo de processamento da ontologia e no carregamento de um ficheiro que contém os axiomas da ontologias para o XSB.

Para fazer a comparação da eficiência do raciocinador do NoHR desktop com o raciocinador do NoHR Web foram testadas várias ontologias usando os raciocinadores ELK, HermiT e Konclude, em que foram comparados os tempos de processamento da ontologia e o tempo que o XSB demora a carregar o ficheiro.

Na figura 5.1 é possível analisar os tempos de inicialização de um raciocinador do NoHR em diferentes ontologias usando o raciocinador ELK. A inicialização do raciocinador foi dividida no processamento da ontologia, a cinzento, e processamento do ficheiro que contém os axiomas pelo XSB, a azul, de forma a ser possível analisar os tempos das diferentes tarefas na inicialização do raciocinador. Para estes testes não foram utilizadas regras associadas a nenhuma ontologia, visto que as regras apenas aumentariam o tempo de tradução, ou seja, no processamento de forma linear, como referido em [14], sendo válido tanto para o NoHR *desktop* como para o NoHR Web.

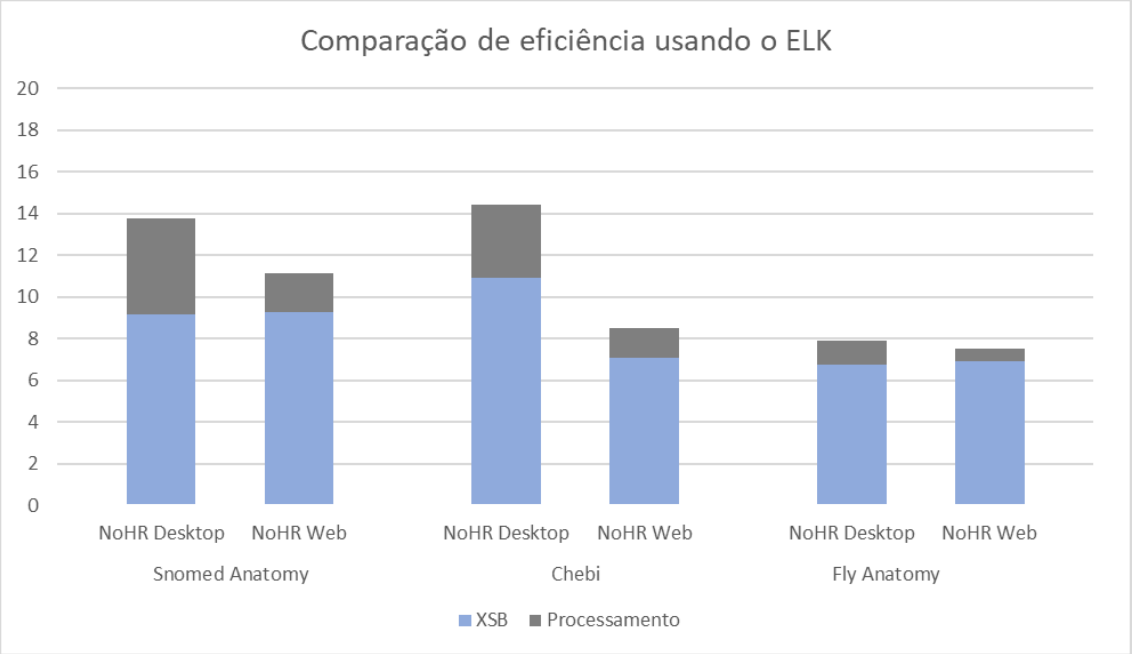

Figura 5.1: Comparação do tempo de pré-processamento usando o raciocinador ELK entre o NoHR desktop e o NoHR Web em segundos

Apesar de terem sido testadas ontologias relativamente pequenas na figura 5.1, é possível perceber que o NoHR Web teve melhores tempos de processamento da ontologia que o NoHR *desktop*.

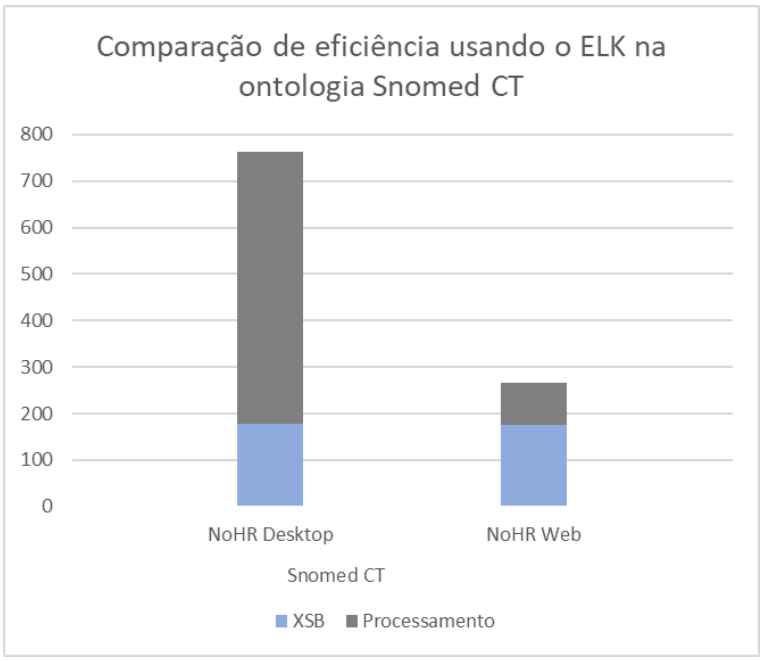

Figura 5.2: Comparação da eficiência na ontologia Snomed CT do raciocinador ELK entre o NoHR desktop e o NoHR Web em segundos

Na figura 5.2, foi comparado a eficiência das duas aplicações numa ontologia bastante

grande como a ontologia Snomed CT no raciocinador ELK. Pela figura é possível perceber que embora os tempos do XSB sejam similares, existe uma grande diferença no tempo de processamento da ontologia. O NoHR Web foi cerca de 7 vezes mais rápido a processar a ontologia Snomed CT do que o NoHR *desktop*, o que é uma diferença bastante significativa.

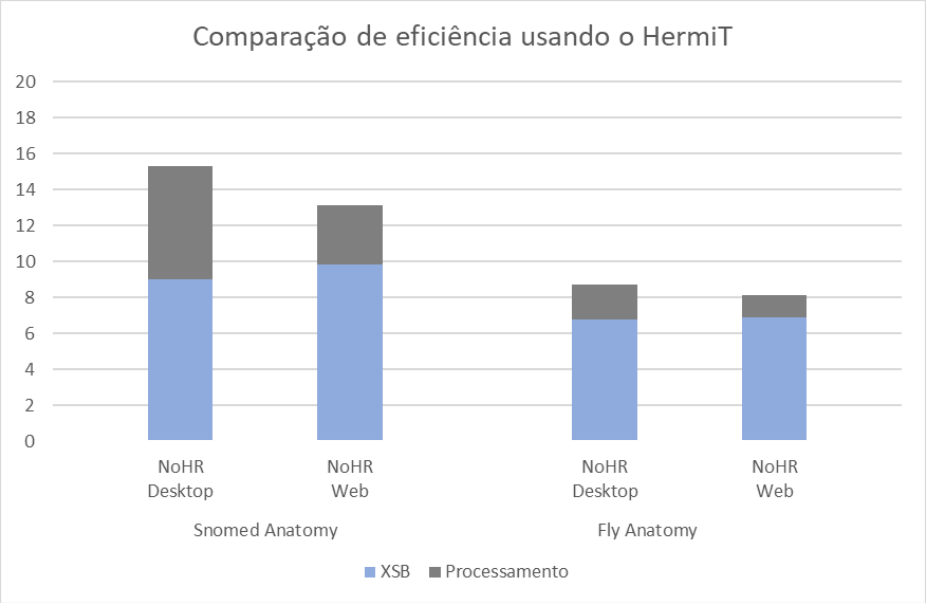

Figura 5.3: Comparação do tempo de pré-processamento usando o raciocinador HermiT entre o NoHR desktop e o NoHR Web em segundos

Na figura 5.3, foi comparado o tempo de inicialização do raciocinador HermiT no NoHR *desktop* e no NoHR Web. É possível perceber mais uma vez que os tempos do XSB são bastante similares embora os tempos de processamento sejam inferiores no NoHR Web.

Na figura 5.4, foi comparado o tempo de inicialização do raciocinador Konclude em ambas as aplicações. É possível analisar o mesmo padrão de tempos já mencionado nos outros raciocinadores, em que o tempo de processamento é inferior no NoHR Web. É possível perceber que quanto maior a ontologia maior a diferença do tempo de processamento do NoHR *dektop* e do NoHR Web.

É possível observar que o ELK tem um tempo menor de processamento comparativamente ao *HermiT* e ao *Konclude* para estas ontologias testadas, confirmando assim o que foi afirmado anteriormente para ontologias pequenas ou médias.

Esta diferença do tempo de processamento da ontologia pelos diferentes raciocinadores, com vantagem para o NoHR Web, deve-se ao facto da eficiência do processamento do NoHR *desktop* estar a ser limitada pelo *Protégé*, visto que nas versões mais recentes do *Protégé*, os tempos de processamento no NoHR *desktop* ainda são maiores, e por isso foi usada uma versão mais antiga do *Protégé* para efetuar os testes no NoHR *desktop*.

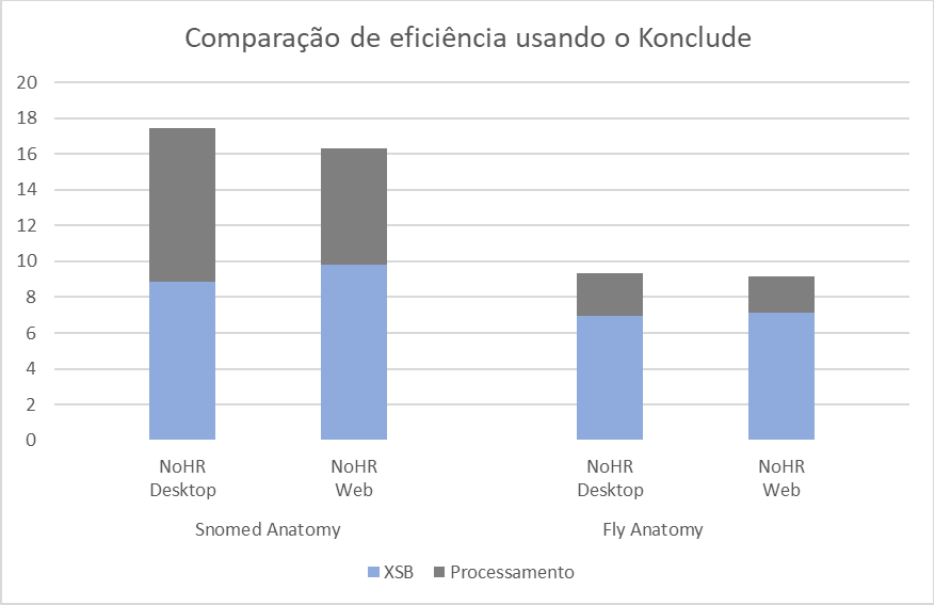

Figura 5.4: Comparação do tempo de pré-processamento usando o raciocinador Konclude entre o NoHR desktop e o NoHR Web em segundos

5.5 Avaliação de desempenho em utilização simultânea

Os testes anteriores, apesar de nos darem informação acerca da eficiência do raciocinador de ambas as versões, apenas testa o NoHR Web com um utilizador e dessa forma não avalia as condições reais de utilização do NoHR web.

Estando o poder de raciocínio do NoHR web num servidor, é necessário fazer testes à eficiência da inicialização de vários raciocinadores e o tempo de processamento de múltiplos raciocínios quando usado por diversos utilizadores em simultâneo.

Para a realização destes testes foi usado uma *framework*, denominada Selenium[1], que permite efetuar testes funcionais em aplicações Web. o Selenium fornece um conjunto de bibliotecas e permite escrever testes nas linguagens *C#*, *Groovy*, *Java*, *Perl*, *PHP*, *Python*, *Ruby and Scala*. No nosso caso, foi usada a linguagem Java, em que foi criado um *script* de forma a automatizar o percurso de um utilizador pelo NoHR Web usando um *browser*.

Para a realização deste teste foram usadas duas máquinas, mencionadas na secção 5.1, uma para fazer pedidos de vários utilizadores e a outra como servidor. A utilização de duas máquinas, deveu-se ao facto de não querer adicionar uma carga suplementar nos recursos do servidor entre os pedidos de vários clientes e as tarefas de servidor, dessa forma os testes não seriam conclusivos da eficiência real do NoHR Web.

Para fazer o teste com múltiplos utilizadores, foram criadas várias *threads*, uma *thread* para cada utilizador, e em cada uma foi executado um *script* que faz o pedido para a inicialização do raciocinador do NoHR. Todos os utilizadores foram registados previamente, em que foi criado um projeto com a mesma ontologia em todos eles.

[1] https://www.selenium.dev/

Posteriormente, de forma a que todas as *threads* executassem o pedido para a inicialização do raciocinador ao mesmo tempo, as mesmas foram inicializadas e colocadas em espera antes da operação de submissão do pedido de inicialização do raciocinador. Só quando todas as *threads* tivessem inicializadas e neste estado é que foram retomadas, e todas efetuavam o pedido praticamente em simultâneo. Foi utilizada esta estratégia, por não ser possível controlar a velocidade a que cada uma executava o *script*, apesar delas serem inicializadas todas praticamente ao mesmo tempo, podia acontecer que algumas *threads* começassem a executar o pedido de inicialização do raciocinador um tempo considerável antes das outras, e isto não permitiria tirar resultados conclusivos dos tempos dos vários utilizadores com a sobrecarga dos restantes.

Foram criados vários utilizadores com as ontologias Chebi e Fly Anatomy e foram feitos testes com 5, 10 e 25 utilizadores a inicializarem um instância do raciocinador do NoHR em simultâneo. Foram analisados os tempos de cada um, tanto no processamento da ontologia como no carregamento do ficheiro gerado para o XSB e foi feita a média dos tempos que cada utilizador demorou a inicializar uma instância do raciocinador.

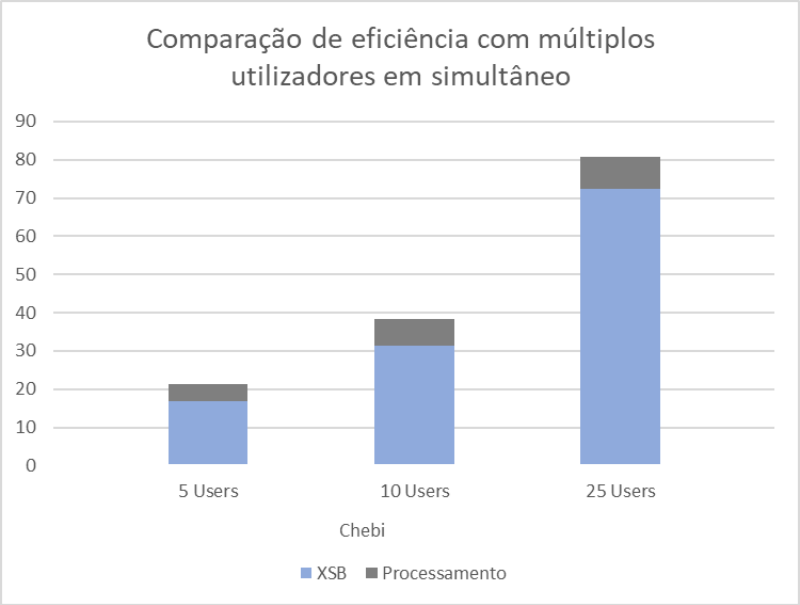

Figura 5.5: Comparação do tempo de pré-processamento usando o raciocinador ELK no NoHR Web com múltiplos utilizadores com a ontologia Chebi em segundos

O gráfico da figura 5.5 mostra o tempo médio que as tarefas de processamento e de carregamento do ficheiro para o XSB levaram a terminar na ontologia Chebi, enquanto que o gráfico da figura 5.6, mostra a mesma informação relativamente à ontologia Fly Anatomy. É possível perceber que o tempo de processamento da ontologia, em média, não sofreu um grande aumento com o incremento do número de pedidos, embora, no caso do carregamento para o XSB, o mesmo já não se verifique. No caso do XSB é possível observar um incremento do tempo proporcional ao número de pedidos em simultâneo. A memória foi monitorizada durante o teste e não houve um impacto que pudesse explicar este

aumento dos tempos, mas é provável que esteja relacionado com o número de leituras e escritas em simultâneo no disco, visto que cada instância do raciocinador gera um ficheiro para ser carregado no XSB.

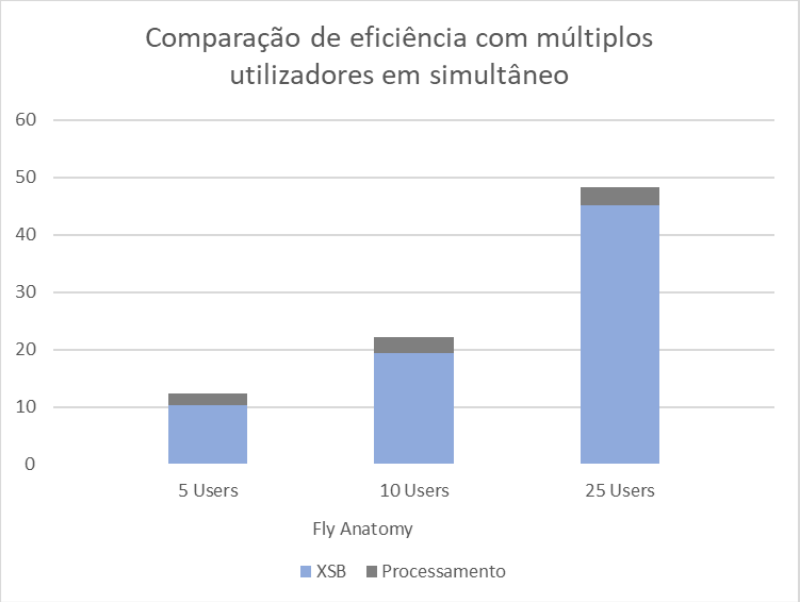

Figura 5.6: Comparação do tempo de pré-processamento usando o raciocinador ELK no NoHR Web com múltiplos utilizadores com a ontologia Fly Anatomy em segundos

5.5.1 Avaliação de desempenho da execução de perguntas no NoHR Web

Para além da comparação dos tempos de inicialização do raciocinador do NoHR Web, foi também testado o impacto da execução de perguntas com múltiplos utilizadores em simultâneo.

Este teste foi feito usando a ontologia *Fly Anatomy*, em que para testar a execução de perguntas foram geradas regras através de um gerador de regras não-monotónicas. Foram geradas regras, com no máximo de 10 átomos no corpo, e factos num rácio de 1:10. Para além das regras e dos factos, foi também criada a mesma quantidade de indivíduos e de regras.

Nº de Regras	1 000
Nº máximo de átomos no corpo das regras	10
Nº de factos	10 000
Nº de diferentes indivíduos	1 000
Nº de novos predicados	1 000
Nº máximo de aridade dos predicados	3

Tabela 5.3: Comparação do tempo de execução de perguntas com diferentes números de utilizadores em simultâneo em segundos

Na tabela 5.3, estão definidos os parâmetros usados para a criação de regras para as ontologias em teste. No caso da ontologia *Chebi*, foram feitos testes com 100 000 factos, mantendo as rácios mencionados anteriormente.

Foram testadas perguntas com 1, 5, 10 e 25 utilizadores em simultâneo e feita a média do tempo de execução de todos os utilizadores.

Ontologias / Utilizadores	**Nº de factos**	**1**	**5**	**10**	**25**
Fly Anatomy	10 000	0.018s	0.021s	0.023s	0.027s
Fly Anatomy	100 000	0.028s	0.035s	0.036s	0.044s
Fly Anatomy	500 000	0.044s	0.048s	0.063s	0.070s

Tabela 5.4: Comparação do tempo de execução de perguntas com diferentes números de utilizadores em simultâneo em segundos

Na tabela 5.4, é possível observar que o tempo de execução de perguntas não é muito afetado com o número de utilizadores em simultâneo, embora seja possível observar um aumento quase negligenciável no tempo de execução de perguntas com o aumento do números de utilizadores e com o aumento do número de factos.

Capítulo 6

Conclusão

Neste capítulo vamos resumir todo o trabalho feito até ao momento e a forma como foi alcançado e deixar algumas sugestões do que pode ainda ser melhorado no futuro.

Como referido no primeiro capítulo, no mundo real existe a necessidade da integração do formalismo de mundo aberto com o formalismo de mundo fechado, ou seja a integração de ontologias com regras não-monotónicas, em que o NoHR por sua vez é a primeira ferramenta que permite esta integração e por isso existe uma grande necessidade de continuar a evoluir este projeto no futuro.

Com o objetivo de melhorar o NoHR e levar as suas funcionalidades para a web, de forma a inovar a forma de trabalhar no *plug-in*, foi criada a aplicação NoHR Web, que foi desenvolvida a partir da aplicação *WebProtégé*.

Uma das limitações do NoHR *desktop*, era a necessidade de instalação e configuração do *plug-in* e do *Protégé*, como também a dificuldade ou impossibilidade de ter projetos partilhados. Com o NoHR Web, estas limitações foram colmatadas, onde neste momento, o utilizador apenas precisa de criar uma conta no NoHR Web para começar a trabalhar com o NoHR. Para além disso, não obriga o utilizador a ter uma máquina que suporte o grande poder computacional exigido pelas tarefas de raciocínio do NoHR. O utilizador pode trabalhar a partir qualquer dispositivo que tenha acesso à Internet, o que é também uma mais valia para o utilizador.

Desta forma, foi criada a primeira aplicação web que permite fazer pesquisas em ontologias que integram regras não-monotónicas.

O desenvolvimento desta aplicação teve vários desafios que tiveram de ser ultrapassados, como a integração das funcionalidades do NoHR num contexto Web e a mudança do paradigma de utilização, onde teve de ser repensada a forma de interação dos vários utilizadores com as funcionalidades NoHR.

Foi criada de raiz uma estrutura que permite associar informações relacionadas com o NoHR aos utilizadores e armazená-las no servidor, mais concretamente, na base de dados *MongoDB*. Como já mencionado, foi aproveitada a utilização que o *WebProtégé* fazia do *MongoDB* para adicionar esta funcionalidade ao NoHR. Desta forma, é menos uma tarefa que o utilizador terá de efetuar no final da sua sessão de trabalho, visto que toda a informação é guardada em tempo real pela aplicação.

Para além da mudança de paradigma, tivemos de nos adaptar à forma como o *WebProtégé* funciona e à *framework* onde o mesmo foi desenvolvido, o *Google Web Toolkit*. Todas as janelas do NoHR foram criadas de raiz para o NoHR Web e com o objetivo de estarem visualmente integradas com as restantes janelas da interface do *WebProtégé*.

No desenvolvimento da interface houve também um grande desafio para perceber que bibliotecas do Java eram autorizadas pela *framework* de forma a serem traduzidas para *JavaScript*, e como podia ser integrada com as funcionalidades do NoHR, onde para isso, foi necessário trabalhar em dois paradigmas diferentes, o cliente e o servidor.

Um dos objetivos era melhorar a interação dos utilizadores com o NoHR. Para além disso, como demonstrado no capítulo de testes, não só melhorou a forma de trabalhar para os utilizadores como melhorou a eficiência de raciocínio do NoHR, o que é bastante benéfico e mais um ponto a favor do NoHR Web.

6.0.1 Trabalho Futuro

Como mencionado anteriormente, é necessário continuar a trabalhar no NoHR, de forma a adicionar novas funcionalidades e continuar a evoluir este projeto. Seria interessante tornar o NoHR Web a versão principal do NoHR, e para isso, migrar o trabalho a realizar no futuro para o NoHR Web, e para isso perceber se o trabalho realizado futuramente no NoHR *desktop* pode ser migrado para o NoHR Web e vice-versa.

A forma como o NoHR Web está estruturado permite que todas as alterações que sejam feitas no raciocinador do NoHR sejam rapidamente passadas para o NoHR Web, visto que foi mantida a estrutura do projeto do NoHR *desktop* para facilitar futuras atualizações.

Devido às limitações impostas pelo *Google Web Toolkit* no desenvolvimento da interface, algumas funcionalidades ficaram do lado do servidor, embora pudessem ficar também no lado do cliente. Para trabalho futuro, era um bom ponto de partida o estudo da melhoria da integração do cliente com o servidor em algumas funcionalidades do NoHR. Por exemplo, o *parser* de regras não-monotónicas poderá dar informações mais precisas dos erros de inserção do utilizador em tempo real, estando no lado do cliente. Estando no lado do servidor, só quando submetida uma regra é que será avaliada pelo *parser* do raciocinador do NoHR. Este tipo de integração tem espaço para ser abordado e melhorado no futuro.

Como trabalho imediato, seria interessante disponibilizar o NoHR Web ao público exterior através de uma máquina virtual que fosse acessível a todos, onde poderá ser propriamente testado num contexto real de utilização.

Num futuro mais a longo prazo, seria interessante estender o NoHR para a *Linked Open Data*, que de forma resumida, permite fazer a ligação de vários conceitos entre as diferentes fontes de informação livres na web como as bases de dados externas, facilitando a extensão de modelos de dados. Como resultado, a integração e a pesquisa em informação complexa torna-se mais simples e muito mais eficiente. Dessa forma, seria um projeto bastante interessante na área da web semântica.

Bibliografia

[1] J. J. Alferes, M. Knorr e T. Swift. "Query-Driven Procedures for Hybrid MKNF Knowledge Bases". Em: *ACM Trans. Comput. Log.* 14.2 (2013), 16:1–16:43. DOI: 10.1145/2480759.2480768. URL: https://doi.org/10.1145/2480759.2480768.

[2] G. Antoniou, P. T. Groth, F. van Harmelen e R. Hoekstra. *A Semantic Web Primer, 3rd Edition*. MIT Press, 2012. ISBN: 978-0-262-01828-9.

[3] N. Costa, M. Knorr e J. Leite. "Next Step for NoHR: OWL 2 QL". Em: *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*. Vol. 9366. Lecture Notes in Computer Science. Springer, 2015, pp. 569–586. ISBN: 978-3-319-25006-9. DOI: 10.1007/978-3-319-25007-6. URL: https://doi.org/10.1007/978-3-319-25007-6.

[4] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer e H. Tompits. "Combining answer set programming with description logics for the Semantic Web". Em: *Artif. Intell.* 172.12-13 (2008), pp. 1495–1539. DOI: 10.1016/j.artint.2008.04.002. URL: https://doi.org/10.1016/j.artint.2008.04.002.

[5] A. V. Gelder, K. A. Ross e J. S. Schlipf. "The Well-Founded Semantics for General Logic Programs". Em: *J. ACM* 38.3 (1991), pp. 620–650. DOI: 10.1145/116825.116838. URL: https://doi.org/10.1145/116825.116838.

[6] M. Gelfond e V. Lifschitz. "The Stable Model Semantics for Logic Programming". Em: *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*. 1988, pp. 1070–1080. ISBN: 0-262-61056-6.

[7] B. Glimm, I. Horrocks, B. Motik, G. Stoilos e Z. Wang. "HermiT: An OWL 2 Reasoner". Em: *J. Autom. Reasoning* 53.3 (2014), pp. 245–269. DOI: 10.1007/s10817-014-9305-1. URL: https://doi.org/10.1007/s10817-014-9305-1.

[8] B. Grosof, I. Horrocks, R. Volz e S. Decker. "Description Logic Programs: Combining Logic Programs with Description Logic". Em: *Description Logic Programs: Combining Logic Programs with Description Logic* (abr. de 2003).

[9] V. Haarslev e R. Möller. "RACER System Description". Em: *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*. 2001, pp. 701–706. ISBN: 3-540-42254-4. DOI: 10.1007/3-540-45744-5. URL: https://doi.org/10.1007/3-540-45744-5.

[10] P. Hayes, ed. *RDF Semantics*. Available at https://www.w3.org/TR/2004/REC-rdf-mt-20040210/. W3C Recommendation 10 February 2004, 2004.

[11] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider e S. Rudolph. *OWL 2 Web Ontology Language Primer*. W3C Recommendation. World Wide Web Consortium, 2009. URL: http://www.w3.org/TR/owl2-primer/.

[12] I. Horrocks e P. F. Patel-Schneider. "A proposal for an owl rules language". Em: *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*. ACM, 2004, pp. 723–731. ISBN: 1-58113-844-X.

[13] I. Horrocks, O. Kutz e U. Sattler. "The Even More Irresistible SROIQ". Em: *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*. AAAI Press, 2006, pp. 57–67. ISBN: 978-1-57735-271-6.

[14] V. Ivanov, M. Knorr e J. Leite. "A Query Tool for EL with Non-monotonic Rules." Em: *International Semantic Web Conference (1)*. Ed. por H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty e K. Janowicz. Vol. 8218. Lecture Notes in Computer Science. Springer, 2013, pp. 216–231. ISBN: 978-3-642-41334-6. URL: http://dblp.uni-trier.de/db/conf/semweb/iswc2013-1.html#IvanovKL13a.

[15] V. Ivanov, M. Knorr e J. Leite. "A Query Tool for *EL* with Non-monotonic Rules". Em: *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*. Vol. 8218. Lecture Notes in Computer Science. Springer, 2013, pp. 216–231. ISBN: 978-3-642-41334-6. DOI: 10.1007/978-3-642-41335-3. URL: https://doi.org/10.1007/978-3-642-41335-3.

[16] V. Kasalica. "NoHR: Integrating Rules and Ontologies with External Data Sets". Tese de mestrado. 2829-516 Caparica: Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa, 2017.

[17] Y. Kazakov, M. Krötzsch e F. Simancik. "The Incredible ELK - From Polynomial Procedures to Efficient Reasoning with $\mathscr{EL}$ Ontologies". Em: *J. Autom. Reasoning* 53.1 (2014), pp. 1–61. DOI: 10.1007/s10817-013-9296-3. URL: https://doi.org/10.1007/s10817-013-9296-3.

[18] M. Kifer e H. Boley, eds. *RIF Overview (Second Edition)*. Available at http://www.w3.org/TR/rif-overview/. W3C Working Group Note 5 February 2013, 2013.

[19] M. Knorr, J. J. Alferes e P. Hitzler. "Local closed world reasoning with description logics under the well-founded semantics". Em: *Artif. Intell.* 175.9-10 (2011), pp. 1528–1554. DOI: 10.1016/j.artint.2011.01.007. URL: https://doi.org/10.1016/j.artint.2011.01.007.

[20] V. Lifschitz. "Nonmonotonic Databases and Epistemic Queries". Em: *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*. Morgan Kaufmann, 1991, pp. 381–386. ISBN: 1-55860-160-0. URL: http://ijcai.org/proceedings/1991-1.

[21] C. Lopes, M. Knorr e J. Leite. "NoHR: Integrating XSB Prolog with the OWL 2 Profiles and Beyond". Em: vol. 10377. Lecture Notes in Computer Science. Springer, 2017. ISBN: 978-3-319-61659-9. DOI: 10.1007/978-3-319-61660-5. URL: https://doi.org/10.1007/978-3-319-61660-5.

[22] D. L. McGuinness e F. van Harmelen, eds. *OWL Web Ontology Language Overview*. Available at https://www.w3.org/TR/owl-features/. W3C Recommendation 10 February 2004, 2004.

[23] A. Miles e S. Bechhofer, eds. *SKOS Simple Knowledge Organization System Reference*. Available at https://www.w3.org/TR/skos-reference/. W3C Recommendation 18 August 2009, 2009.

[24] B. Motik e R. Rosati. "Reconciling description logics and rules". Em: *J. ACM* 57.5 (2010), 30:1–30:62. DOI: 10.1145/1754399.1754403. URL: https://doi.org/10.1145/1754399.1754403.

[25] B. Motik, P. F. Patel-Schneider e B. C. Grau, eds. *OWL 2 Web Ontology Language Direct Semantics (Second Edition)*. Available at http://www.w3.org/TR/owl2-direct-semantics/. W3C Recommendation 11 December 2012, 2012.

[26] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue e C. Lutz, eds. *OWL 2 Web Ontology Language Profiles (Second Edition)*. Available at https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/. W3C Recommendation 11 December 2012, 2012.

[27] C. Patel, J. J. Cimino, J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, L. Ma, E. Schonberg e K. Srinivas. "Matching Patient Records to Clinical Trials Using Ontologies". Em: *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*. Vol. 4825. Lecture Notes in Computer Science. Springer, 2007, pp. 816–829. ISBN: 978-3-540-76297-3. DOI: 10.1007/978-3-540-76298-0. URL: https://doi.org/10.1007/978-3-540-76298-0.

[28] R. Reiter. "What Should a Database Know?" Em: *J. Log. Program.* 14.1&2 (1992), pp. 127–153. DOI: 10.1016/0743-1066(92)90049-9. URL: https://doi.org/10.1016/0743-1066(92)90049-9.

[29] M. Schneider, ed. *OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition)*. Available at https://www.w3.org/TR/owl2-rdf-based-semantics/. W3C Recommendation 11 December 2012, 2012.

[30] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur e Y. Katz. "Pellet: A practical OWL-DL reasoner". Em: *J. Web Semant.* 5.2 (2007), pp. 51–53. DOI: 10.1016/j.websem.2007.03.004. URL: https://doi.org/10.1016/j.websem.2007.03.004.

[31] A. Steigmiller, T. Liebig e B. Glimm. "Konclude: System description". Em: *J. Web Semant.* 27-28 (2014), pp. 78–85. DOI: 10.1016/j.websem.2014.06.003. URL: https://doi.org/10.1016/j.websem.2014.06.003.

[32] D. Tsarkov e I. Horrocks. "FaCT++ Description Logic Reasoner: System Description". Em: *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*. 2006, pp. 292–297. ISBN: 3-540-37187-7. DOI: 10.1007/11814771. URL: https://doi.org/10.1007/11814771.

[33] T. Tudorache, C. Nyulas, N. F. Noy e M. A. Musen. "WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the Web". Em: *Semantic Web* 4.1 (2013), pp. 89–99. DOI: 10.3233/SW-2012-0057. URL: https://doi.org/10.3233/SW-2012-0057.

[34] M. Y. Vardi. "Why is Modal Logic So Robustly Decidable?" Em: *Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop 1996, Princeton, New Jersey, USA, January 14-17, 1996*. Vol. 31. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1996, pp. 149–184. ISBN: 0-8218-0517-7. URL: http://dimacs.rutgers.edu/Volumes/Vol31.html.